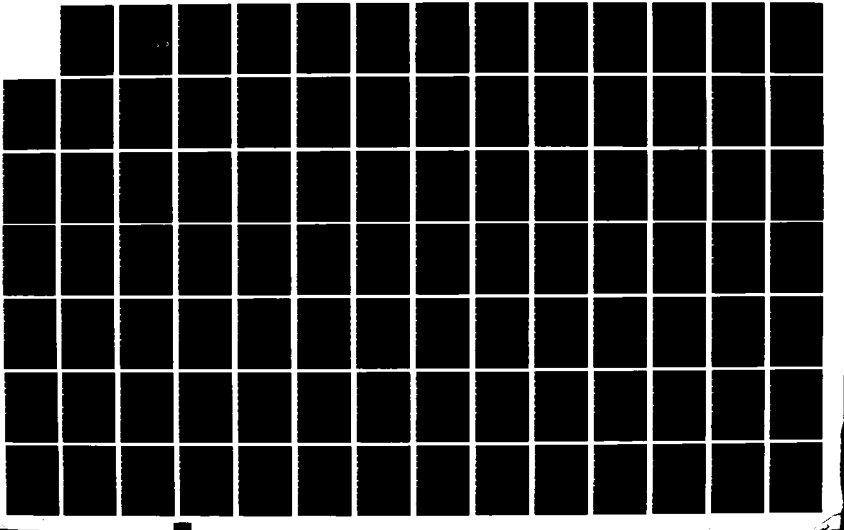
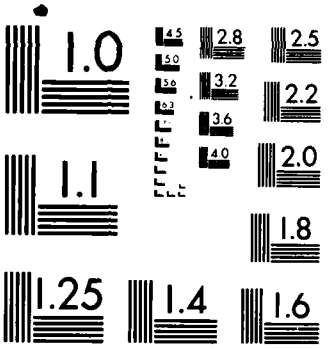
AD-A162 101 DATABASE DESIGN METHODOLOGY AND DATABASE MANAGEMENT SYSTEM FOR COMPUTER-A (U) IOWA UNIV IOWA CITY APPLIED-OPTIMAL DESIGN LAB T S MURTHY ET AL DEC 84 1/2

UNCLASSIFIED CAD-SS-84-20 AFOSR-TR-85-1071 AFOSR-82-0322 F/G 5/1 NL





(2)

Technical Report No. CAD-SS-84.20

Database Design Methodology and Database Management System for Computer-Aided Structural Design Optimization

By

T. Sreekanta Murthy and J. S. Arora

Applied-Optimal Design Laboratory
College of Engineering
The University of Iowa
Iowa City, Iowa 52242

Prepared for the
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Under Grant No. AFOSR 82-0322

December 1984

Technical Report No. CAD-SS-84.20

DATABASE DESIGN METHODOLOGY AND DATABASE MANAGEMENT SYSTEM FOR COMPUTER-AIDED STRUCTURAL DESIGN OPTIMIZATION

T. Sreekanta Murthy and J.S. Arora

Applied-Optimal Design Laboratory
College of Engineering
The University of Iowa
Iowa City, Iowa 52242

Prepared for the
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Under Grant No. AFOSR 82-0322

December 1984

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS: None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; Unlimited distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CAD-SS-84-20	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 85-1071
6a. NAME OF PERFORMING ORGANIZATION: Applied-Optimal Design Laboratory; 6b. OFFICE SYMBOL (If applicable): ADL	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NA
6c. ADDRESS (City, State and ZIP Code): College of Engineering, The University of Iowa, Iowa City, Ia 52242	7b. ADDRESS (City, State and ZIP Code): Bolling AFB, D.C. 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Air Force Office of Scientific Research; 8b. OFFICE SYMBOL (If applicable): NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: Grant No. AFOSR 82-0322
8c. ADDRESS (City, State and ZIP Code): Aerospace Sciences, Bolling AFB, D.C. 20332-6448	10. SOURCE OF FUNDING NOS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2307; TASK NO B1; WORK UNIT NO
11. TITLE (Include Security Classification): Database Design Methodology & Database Management System for	
12. PERSONAL AUTHOR(S): T. SreekantaMurthy and J.S. Arora	
13a. TYPE OF REPORT: Interim; 13b. TIME COVERED: FROM 7-84 TO 12-84	14. DATE OF REPORT (Yr., Mo., Day): 1984-12-31; 15. PAGE COUNT: 147
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR	Database Design, Structural Analysis, Optimization, Database Management System

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A study is made to integrate finite element-based-optimal structural design methods and computer-science methods into a computer-based system containing a database, a program library and man-machine communication link. Emphasis is placed upon database management concepts for structural design. Important components required to build a computer-aided structural design system are described. A number of database management concepts -- hierarchical, network and relational data models, conceptual, internal and external view of data organization, normalization of data, and integrity of database are discussed with reference to structural design data. A methodology to design a database is proposed. Three levels of data organization-conceptual, internal and external are suggested. A methodology to construct a numerical data model is described. A numerical data model supports data of various types of large matrices such as banded, skyline and hyper matrices. Requirements of database management system and components needed to develop it are discussed. Languages required to enable good communication link between designer and computer are

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☒ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. A.K. Amos	22b. TELEPHONE NUMBER (Include Area Code): 202-767-4937; 22c. OFFICE SYMBOL: NA

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

TABLE OF CONTENTS

LIST OF SYMBOLS

$\bar{A}$	Element cross-sectional property matrix
B	Strain-displacement matrix
b, b_i	Design variables
C	Damping matrix of a structure
D	Constitutive matrix
F	Body force
F_e	Element forces
$F^{t+\Delta t}$	Vector of nodal forces equivalent to element stresses
h	Equilibrium equation
K_{bb}	Assembled boundary stiffness matrix
K	Assembled stiffness matrix of a structure
K_b	Condensed stiffness matrix of a substructure
K_{bi}	Assembled internal-boundary stiffness matrix
K_e	Element stiffness matrix
K_{eff}	Effective stiffness matrix of a structure
K_{ii}	Assembled internal stiffness matrix
K_{NL}	Nonlinear geometric stiffness matrix
K_L^t	Linear incremental stiffness matrix
K_σ	Geometric Stiffness matrix
M	Assembled mass matrix of a structure
M_e	Element mass matrix
N	Shape function matrix
n	Element number
P	Global loads on a structure
Φ_B	Element body force

P_b	Boundary loads of a substructure
P_b^*	Condensed load matrix of a substructure
p_e	Element load vector
P_{eff}	Effective load matrix of a structure
P_i	Internal loads of a substructure
P_s	Element surface loads
$P^{t+\Delta t}$	Vector of externally applied loads at time $t+\Delta t$
p_o	Element load due to initial strain
r	Substructure number
T	Transformation matrix
u_e	Element Displacements
U_i	Internal displacements of a structure
U_b	Boundary displacements of a structure
V	Volume
V	Global displacement of a structure
V_b	Boundary displacements of a substructure
V_e	Element displacements
V_i	Internal displacements of a substructure
y	Mode shape
z	Displacement vector
Z_j^a	Allowable displacement at jth location
Z_j	Displacement of j^{th} location
ΔV	Vector of incremental nodal displacements
ε_e	Element strain
ε_o	Initial strain
ρ	Material density
σ_a	Allowable stress in the member

σ_b	Buckling stress of a member
$\boldsymbol{\sigma}_e$	Element stress
σ_{ij}	Stress in the member
λ	Eigen value
$\boldsymbol{\lambda}_i$	Adjoint matrix
$\boldsymbol{\psi}$	Constraint function
ψ_o	Cost function
ζ	Frequency
ζ_o	Lower bound on eigen value

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A study is made to integrate finite element-based-optimal structural design methods and computer-science methods into a computer-based system containing a database, a program library and man-machine communication link. Emphasis is placed upon database management concepts for structural design. Important components required to build a computer-aided structural design system are described. A number of database management concepts -- hierarchical, network and relational data models, conceptual, internal and external view of data organization, normalization of data, and integrity of database are discussed with reference to structural design data. A methodology to design a database is proposed. Three levels of data organization-conceptual, internal and external are suggested. A methodology to construct a numerical data model is described. A numerical data model supports data of various types of large matrices such as banded, skyline and hyper matrices. Requirements of database management system and components needed to develop it are discussed. Languages required to enable good communication link between designer and computer are developed. A database management system - MIDAS is implemented for use in structural design applications. MIDAS supports both relational and numerical data models. It can be used either through application program calls or interactively. Finally, it is concluded that with the proposed database design methodology and the advanced database management system, optimum design of complex structural systems can be attempted.

1. INTRODUCTION

1.1 Introductory Remarks

Advances in computer technology have brought about profound changes in the way engineering analysis and design are performed. In structural analysis computer has become a vital adjunct to theory. Several general purpose computer programs having a wide range of capabilities are being commonly used for finite element analysis of structures. In the structural design field, computer programs are being developed for optimal design. But, they are in the early stage of development and are facing a number of problems in design of practical structures. Problems arise due to iterative nature of optimal design algorithms and need for reanalysis of the structure in each iteration. Reanalysis of a structure using existing finite element programs are difficult because they is not flexible to use modified data generated at various design stages. Moreover, designer needs control over the program and data in selecting appropriate algorithms and data to obtain optimum solution. Thus, there exists a wide gap between structural analysis and design capabilities. To bridge this wide gap, it is necessary to establish approaches through integration of computers in the design environment. The term computer-aided design has evolved over years which provide a good basis for such an integration.

Computer-aided design (CAD) means integration of engineering methods and computer-science in a computer-based system, providing a database, a program library and a man-machine communication link. The term computer-aided structural design optimization is derived from the above definition to cover structural analysis and design optimization methods. In this study a new concept is presented for integrating structural analysis and design optimization methodology into a computer-based systems which encompases this meaning of CAD. Emphasis is placed upon database management concepts for finite element analysis and structural design optimization.

1.2 Computers in Structural Design - State-of-the-Art

Knowledge about the historical background provides a better understanding of the state-of-art. Going back to 1960, Integrated Civil Engineering Systems (ICES) development was an important milestone in the use of computers in solving civil engineering problems. It was based on the idea of integrating computers in the problem solving environment to provide faster, more accurate and complete analysis and design capability to engineers. During the same period, theory of the finite element method began to evolve and led to the development of several finite element analysis programs. The computer programs such as NASTRAN, STRUDL, and ASKA are well known and widely used for finite element analysis. Many of these programs are quite sophisticated, large in size and are capable of analyzing a wide variety of structural problems. With the advances in computer technology, computers are available at a lesser cost and have additional facilities like graphic display, and large disk capacities. Finite element programs were designed to make use of such facilities to display finite element mesh, store large amounts of data on disk and provide interactive facility to users. Programs like GIFTS, ANSYS, and ADINA were developed in the seventies to provide these new features to users.

During the last decade, research activity in the area of structural design optimization increased. Investigations on nonlinear programming techniques in structural optimization became one of the major topics of research. Several computer programs were developed for solving structural optimization problems. These include DOCS (Arora, et al., 1984a), ACCESS (Fleury, et al., 1981), PROSSS (Sobieszczanski-Sobieski, et al., 1980) ODYSSEY (Bennett, 1979) and others having moderate range of capabilities in solving optimization problems. They use finite element method for analysis of structures. DOCS program has capability to use substructuring, design damaged structures, and to use gradient-based techniques for optimization. PROSSS uses SPAR program for finite element analysis. Many of these programs were developed to study problems of research interest. Therefore, applicability of the programs to general structural problems is limited. None of these programs is linked to any pre- or post-processor making input to program and analysis of results extremely difficult. Studies are being made to develop good structural design optimization programs that are comparable to generality and capabilities of existing finite element programs.

Since, finite element analysis of structures uses large amount of data, some routines were incorporated into finite element packages to store data on secondary storage devices. Data management using these routines were tedious and unorganized. Data generated by finite element packages were almost impossible to use in other programs for further analysis and design of structures. Development of design optimization algorithms, faced this problem for using analysis data generated by finite element packages. Iterative design process posed a big challenge not only in efficient use of computer resources, but also in organization of large amount of data of finite element analysis and design optimization methods. At this stage, engineering software designers began to think of introducing database management concepts into the software similar to those of business database management systems. Several database management programs were developed in the late seventies and in the beginning of the eighties for engineering applications. Integrated programs for Aerospace Vehicle Design (IPAD) development is an important milestone in engineering database management. A database management system called RIM (RIM, 1982) was developed under IPAD project. Several application programs such as SPAR (Giles and Haftka, 1978) BANDIT (band width minimization), PROSS, ATLAS, and NPLOT (graphics) were tied together with a common database for integrated design of structural systems (Fishwick and Blackburn, 1982). However, use of a database in these application programs were limited to input and output only. There does not exist a finite element program which directly uses a generalized database management system such as the one developed for IPAD project. Studies are being made to incorporate a database, a program library and man-machine communication link as needed for computer-aided structural design. Emphasis on blending computer-science and engineering methodology toward arriving at efficient and economic design of structural systems seems to be the goal of CAD today.

1.3 Motivation for Research

In optimal design of structural systems we generally use nonlinear programming and finite element techniques. Nonlinear programming techniques require formulation of design objectives and constraints of the system. They use large amount of data depending on the size and complexity of problem.

Organization of data related to design variables, geometry, material, loads, and intermediate computation data, generated and used in design of large structural systems is a tedious task. Finite element techniques are usually adopted to analyze the system within a design iteration. As such the finite element techniques require huge amount of computation and data storage depending on the size of problem at hand. Further, the amount of data handled depends directly on the number of iterations performed in iterative design optimization algorithms. Therefore, there is a need for data organization in optimal design of structural systems.

In this regard, incorporating a database into structural design programs look very attractive. Such a database can provide data for both structural design optimization and finite element analysis programs. It will enable designers to choose appropriate data from the database and use them in any optimization algorithms to improve design. Also, data used and generated in one program can be made available for use in another program. Since, most of the data for finite element analysis and design optimization are common, a centralized database will provide efficient organization of computer resources. A centralized database allows interaction between a finite element program and an optimization program to improve design iteratively. Such a database will provide an option for the designer to interrupt the program execution and provide flexibility for the designer to change the design parameters. A good database will enable addition of new optimization and other programs which use the common data without extensive modification of database or existing programs. Also, several designers can be allowed to use a common database to investigate alternate designs. Interactive graphics data can be stored in a database to provide easy communication between computer and designer. Therefore, a properly designed database, together with a set of design programs and communication system, offer a considerable aid to engineers involved in design optimization.

In view of the above observation, we will investigate design and use of a database in structural design optimization.

1.4 A Survey of Literature

A survey of literature of data management in computer-aided structural design is presented in this section. Also, the survey includes literature on database management for engineering applications. The survey is broadly classified into database management concepts and systems. Various database management systems currently in use are reviewed in Chapter 5 and their features tabulated there.

The meaning of computer-aided design has changed several times in the past two decades of its usage. It was a popular idea that CAD meant a menu of analysis programs called by the designer. Later, CAD became synonymous with computer-aided drafting. However, a true description of CAD is synergistic interplay of man and computer (Allan 1972). A more appropriate definition of CAD is given by Encarnacao and Schlechtendahl (1983): "It is a discipline that provide know-how in computer-software and hardware in system analysis and in engineering methodology for specifying, designing, implementing, introducing, and using computer based systems for design purpose."

The paper by Felippa (1979) serves as an introduction to the subject of database management for scientific and engineering applications. The paper highlights the difference between the business data management and scientific data management. A comprehensive list of terminology relevant to scientific computing is given in another paper by the author (Felippa, 1980). Since, the terminology used in business DBMS is fairly new to engineers, this list serves as a starting point for the newcomers. It is interesting to know how scientists actually use their data. The paper by Bell (1982) discusses some issues about data usage and also gives comparison between data modelling for scientific and business applications. In order to bring out difference between the use of database for business and engineering applications, Foisseau and Valette (1982) present a list of criteria.

The application of data management in finite element analysis and design optimization computation is fairly new. Even though, data management in these computations is critically needed, not much attention has been paid to develop proper data management techniques. Only a few research studies were made on data management for finite element analysis. Lopez (1978) and associates studied the application of data management to structures. They pointed out that serious drawbacks of ASKA, STRUDL and NASTRAN were due to lack of high level database management facility, Also, these programs did not provide any type of data structure capability. They have simple internal organization and require many files with trivial data structures. This type of systems tends to be I/O bound because logical operations on data are related directly to a physical location on a serial device. For example, in generating the stiffness matrix for the elements of a structure, most programs generate one matrix and write onto a sequential device; and the process is repeated for all elements of a structure. In order to access these stiffness matrices at a later time, the program must pass serially over the entire file again. Lopez (1974) in another paper presented a data management system for finite element analysis. Pahl (1981) described the properties and functions of data storage for finite element programs.

Several research studies were made on data management techniques for computer-aided design and general engineering applications. The techniques developed for them are also applicable for finite element analysis and design optimization. Studies on data models, database design methodology, database network, data definition language, data manipulation language, database integrity and consistency and numerical database management were conducted by several researchers. A comprehensive survey of data management in engineering applications is given by Sreekanta Murthy and Arora (1984).

The well-known data models -- hierarchical, network and relational have been studied by many researchers to find out their suitability for organizing engineering data. Koriba (1983) discusses applicability of ANSI/SPARC, CODASYL and relational approach to CAD software design. The three levels of data view proposed by ANSI/SPARC is gaining wide acceptance and is likely to be incorporated into future CAD systems. Relational approach is based on set concepts and provide a sound mathematical background. This approach provides high level of data independence, user friendly data definition and data manipulation capabilities. Relational model is becoming popular among database designers and users. Several researchers are currently working on this model. Fishwick and Blackburn (1982) discuss advantage and disadvantage of a relational model from an engineering point of view. Authors provide

examples of relations for managing data of a finite element model. They also described the development of the PRIDE systems which integrates engineering application programs -- AD-2000, SPAR, PROSSS, NCAR, and BANDIT. SPAR and PROSSS are finite element analysis and structural optimization packages respectively. AD-2000 is a finite element model generator and BANDIT is bandwidth optimization program. However, use of their database management system was limited to interfacing application program input and output to a common database. Modification of application programs was not made to use the database for programs internal data organization needs. Blackburn, Storaasli and Fulton (1982) in another paper demonstrate the use of a relational database in engineering applications. Four sample problems -- a panel with circular hole, a square plate, a conventional wing structure and a large area space structure were used to evaluate the merits of managing engineering data using a relational system. Studies on hierarchical model are mainly with reference to organizing large matrices. Lopez (1974) use a hierarchical model for finite element data organization. A hierarchical data structure for organizing node, element, load and stiffness matrix data is given in the paper. However, the DBMS uses a problem-oriented language translating facilities in the POLO supervisor. Under which the application programs operate. Hence, it is highly doubtful that two will ever be used independently of each other. Pahl (1981) described hierarchical storage structure for hypermatrix data organization. Hypermatrix stiffness and load data are found to be most suitable to hierarchical data representation. A paper by Elliot, Kunni, and Browne (1978) describe a hierarchical model of data and a DBMS system design based on it. Some practical examples on structural design and wind tunnel data management are also given in the paper. However, this system require a precompiler to decode the data description and data manipulation commands in a source program.

Investigations have been conducted to find out a suitable way to design a database for engineering applications. There exists basically two different approaches to database design -- first approach generates a global schema and then derive local views from it; the second one obtains local views of different users and then integrate then to form a global view. Buchmann and Dale (1979) analyze different methodologies to database design and present a frame work for evaluating them. A comprehensive description of database design methodology for business applications is given in Vetter and Maddison (1983). Several researchers Lillehagen and Dokkar (1982), Grabowski, Eigener and Ranch (1978), and Eberlein and Wedekind (1982) have worked on database design for CAD applications. However, there do not exist any methodology to design database of finite element and design optimization programs.

Development of suitable data definition (DDL) and data manipulation languages for engineering applications have been of interest to many researchers. One of the major considerations in the design of data definition language was to keep the syntax concise and easy to use for application programmers. Several other important considerations in DDL design are described in detail by Elliot, Kunni and Browne (1978). They use special indicators in the source program code to identify the DDL and DML commands and translate them using a precompiler to FORTRAN statements. These DDL and DML statements can be used to operate on a hierarchical data structure. Special features of DDL and DML in a relational DBMS for interactive design are described by Shenoy and Patnaik (1983).

The application of data management in numerical computation is fairly new. Finite element analysis and design optimization procedure require substantial amount of matrix data processing. Data management system require special facilities to deal with data of large matrices. A recognition of this need is made by Daini (1982) and a model is developed for numerical database arising in many scientific application to keep track of large sparse and dense matrices. The paper presents a generalized facility for providing data independence by relieving users from the need for knowledge of physical data organization on the secondary storage devices. Because of the limitation of core storage and to reduce the input-output operations involved in secondary storage techniques, many investigations have been conducted on the efficient use of primary memory. A detailed survey by Pooch and Nieder (1973) gives various indexing techniques that can be used in dealing with sparse matrices. Darby-Dowman and Mitra (1983) describe a matrix storage scheme in linear programming. Rajan and Bhatti (1983) presented a memory management scheme for finite element software. Sreekanta Murthy, Reddy and Arora (1983) describe the database management concepts that are applicable to design optimization field.

1.5 Objectives of Research

1. To study of various database management concepts applicable to computer-aided structural design optimization field. Suitability of available database management concepts and drawbacks associated with their use in engineering design will be investigated.

2. To develop a suitable database design methodology for structural design database and to develop a conceptual data model to represent the design data. Schemes for constructing external and internal data models will be identified. Data models for organizing matrix data will be developed.

3. To study the existing database management systems and to identify the important features with respect to their suitability to organize structural design data.

4. To propose a suitable data definition language, data manipulation language and query language for engineering design database management system.

5. To implement a database management system for engineering applications based on selected data models.

6. To design a database for finite element analysis based on the suitable model. Use of the database, data definition, data manipulation and query languages will be demonstrated.

7. To design a database for structural design optimization. Use of the database in iterative design data organization, numerical data representation and interactive design process will be demonstrated.

1.6 Scope of Work

Computer-aided structural design process is identified in Chapter 2. Mathematical modelling for finite element analysis and structural design optimization are given. Need for database management in structural design is stressed. Chapter 3 deals with database management concepts. Well-known data models are described with reference to structural design data. Various database management concepts like normalization of data, semantic integrity, and global and local databases are described. Database design methodology for structural design is given in Chapter 4. Methodology for conceptual model development for structural design database is described. Normalization procedures are described with examples. Description of a proposed numerical model is given there. In Chapter 5, components of a database management system are studied. Data definition and data manipulation languages for structural design database are proposed. Considerations in developing memory management schemes and query language are presented. Implementation details of a database management system for organizing structural design data are described in Chapter 6. Relational data management procedures and numerical data management schemes are described. Usefulness and drawbacks of this systems are given. Implementation and evaluation of the database and the database management systems are in progress. Results of that study will be reported at a later date. Finally, discussion and conclusion of the present study are given in the last chapter.

2. COMPUTER-AIDED STRUCTURAL DESIGN

2.1 Introductory Remarks

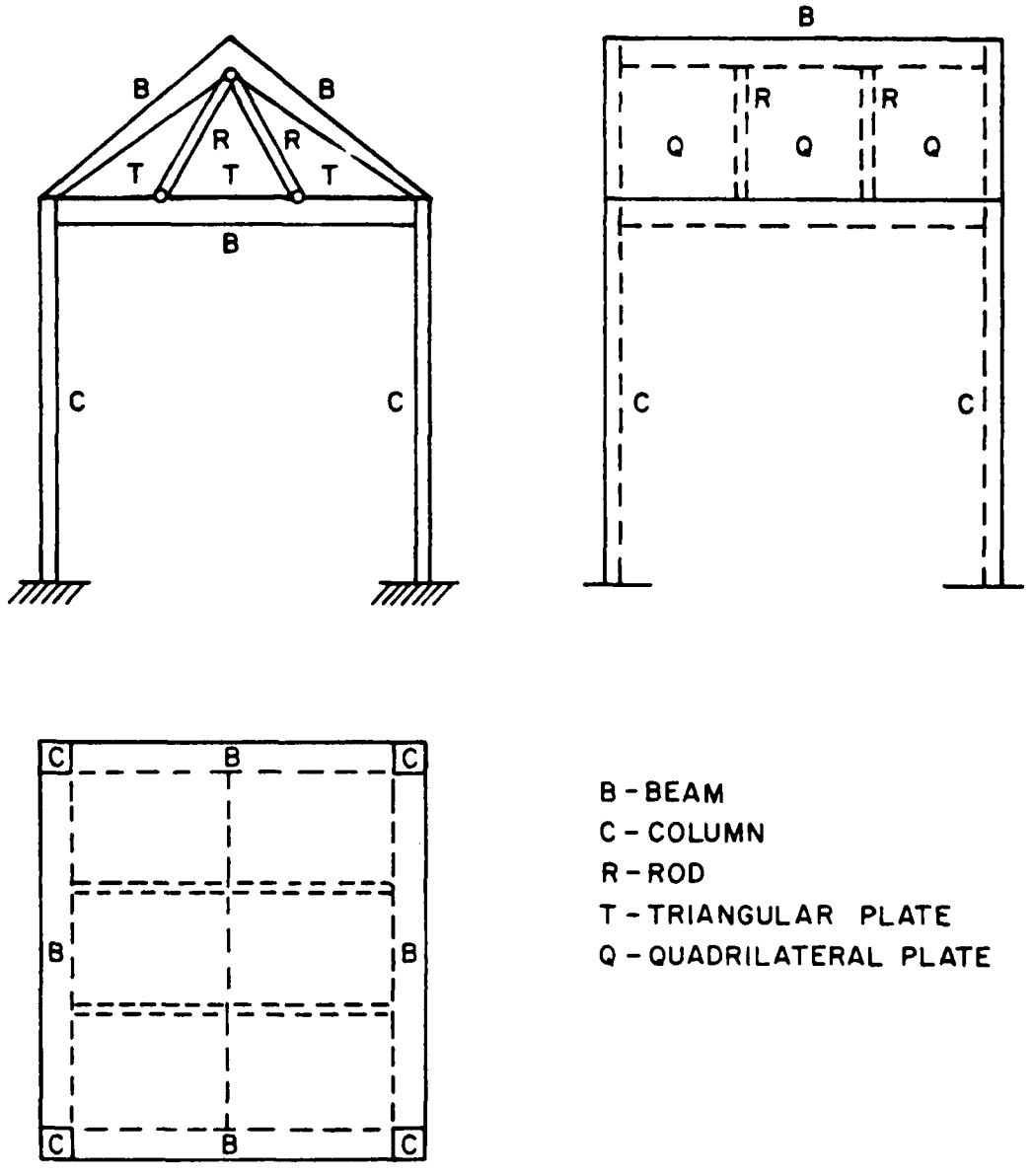
In this section, the principles, methods and tools used for computer-aided design of structural systems are described. Structural design process is described with the aid of a sample design problem to provide qualitative description of the design process. In particular, mathematical modelling of the structural design process is given in Section 2.3 to bring out various steps. As mentioned in Chapter 1, a database, a program library and a communication link form the important components of a CAD system. These components are described in detail in Section 2.4. Need for data management for structural design is emphasized in Section 2.5. Need for a good communication subsystem is given in Section 2.6. Finally, various classes of users of a computer-aided structural design system and their requirements are described in the last section.

2.2 Structural Design Process and a Sample Design Problem

The purpose of studying structural design processes is to provide the CAD system analyst with means of describing the structural system into which CAD must fit. The design process can be described, in general, by a sequence or chains of actions where each action passes its results on to its successors. The complex nature of design process will have to be reflected in CAD systems if such systems are to support the design process as a whole.

The design process begins with the identification of a need by a user of the structural system. The needs and objectives of the system are defined quantitatively. Functional analysis is carried out to find out operational requirements of the structural system. The next step is the configuration or conceptual design of the system. For example, if function to be performed is to support the loads on a frame, the conceptual design (see Fig. 2.2.1) includes beams, columns, plates, and bars. At the conceptual design stage various parameters describing the system are identified and acceptable range of values are prescribed. This preliminary design is then analyzed with respect to the constraints and if it does not adequately satisfy the constraints, the design is revised. This is an optimal design process, which has its objectives the choice of undetermined parameters that were identified in the previous step. The criterion for optimal design may be maximization of structural system capability or minimization of cost. The analysis and redesign cycles are repeated until a design satisfying all the constraints is obtained.

It is common that a number of designers working on a practical design project carry out specific subtasks. Designers are required to meet individual goals, and may have an isolated view of the project. For example, designer A (see Fig. 2.2.2) may work only on part of the structure in the design project. During the process, a set of information required for individual needs is derived to carry out the specific task. The, sharing of information takes place, between the conceptual design level with subsystem design levels, and among ubsystem design levels themselves. The method of information sharing (reports, catalogues, etc. in case of convention design process; a database in case of CAD design process) and tools for information



B - BEAM
C - COLUMN
R - ROD
T - TRIANGULAR PLATE
Q - QUADRILATERAL PLATE

Figure 2.2.1 Conceptual Design of a Frame

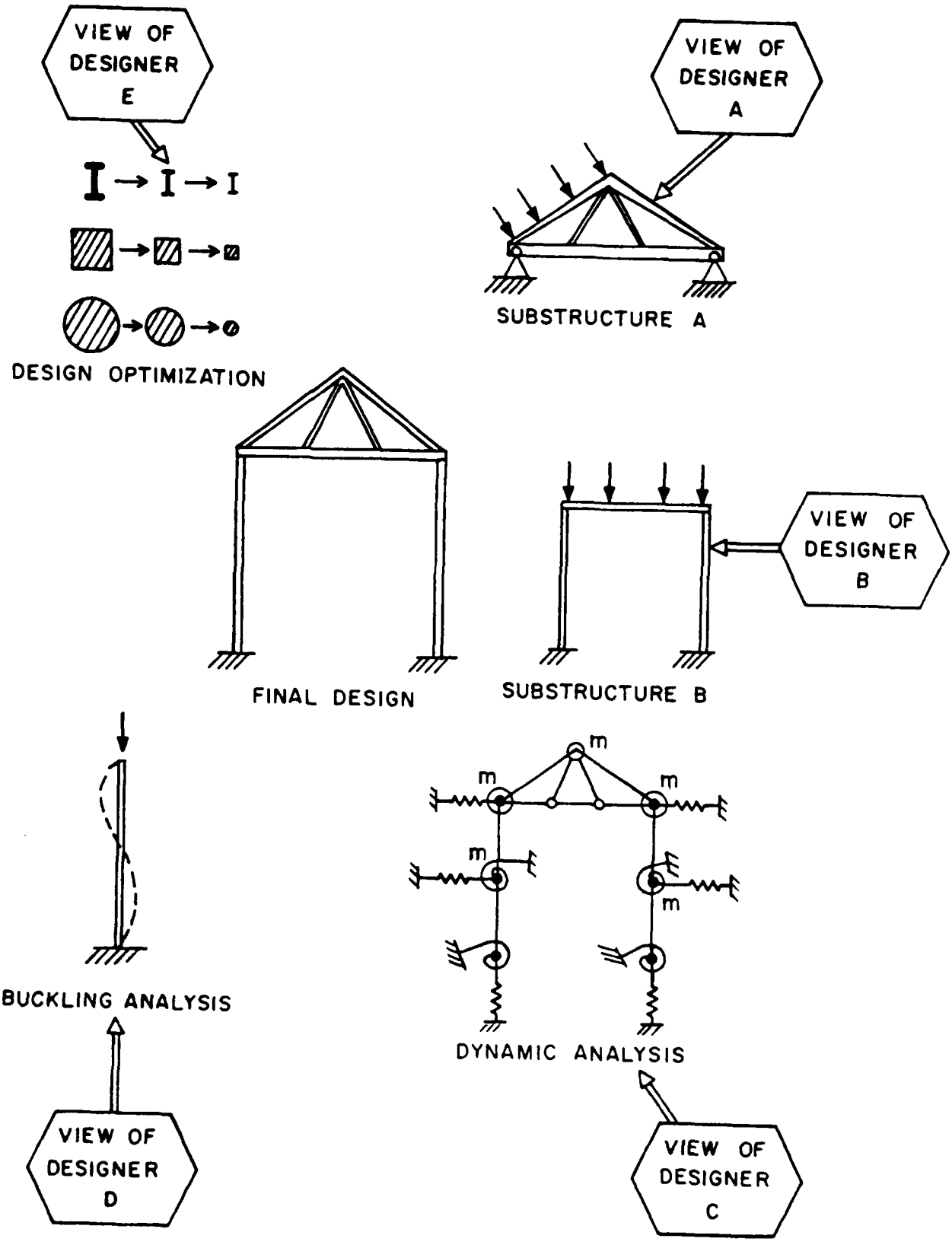


Figure 2.2.2 Designers View of a Frame

sharing (typing, printing, drafting in conventional design; interactive computer terminals, graphics in CAD design process) are dependent on the state-of-art of the design process.

2.3 Mathematical Modelling of Structural Design

In the previous section, a general structural design process was described. Here, mathematical modelling for structural design is given. Various steps of the structural design are formulated in terms of mathematical models. Namely, identification of objectives and constraints, analysis of the structure by finite element method, design constraint checks, design sensitivity calculations and design optimization process are presented. This formulation is intended to provide CAD system analyst the information about the sequence of computation performed, data used in design and its nature.

2.3.1 Finite Element Analysis

Finite element analysis begins with idealization of the structure using a number of finite elements. The input data for a finite element analysis program consists of the geometric idealization, the material properties, and the loading and boundary conditions. Important steps of finite element analysis (Przemieniecki, 1968) are listed below. Depending on the type of analysis, some or combination of the steps are used.

Element Level Computation. At the element level, stiffness matrices, mass matrices, load matrices are computed as

$$\mathbf{K}_e = \int_V \mathbf{B}^T \mathbf{D}\, \mathbf{B}\, dV$$

$$\mathbf{M}_e = \int_V \mathbf{N}^T \rho\, \mathbf{N}\, dV$$

$$\mathbf{P}_e = \mathbf{P}_o + \mathbf{P}_S + \mathbf{P}_B$$

$$\mathbf{P}_o = \int_V \mathbf{N}^T\, \mathbf{D}\, \boldsymbol{\varepsilon}_o\, dV$$

$$\mathbf{P}_S = \int_A \mathbf{N}^T\, \mathbf{P}\, dA$$

$$\mathbf{P}_B = \int_V \mathbf{N}^T\, \mathbf{F}\, dV$$

Substructure Level Computation. If substructures are used in the idealization, then element stiffness matrices are assembled to form substructure level matrices. The equilibrium equation for the r^{th} substructure is given by

$$K^r U^r = P^r$$

i.e.

$$\begin{bmatrix} K_{ii} & K_{bi} \\ K_{ib} & K_{bb} \end{bmatrix}^r \begin{Bmatrix} U_i \\ U_b \end{Bmatrix}^r = \begin{Bmatrix} P_i \\ P_b \end{Bmatrix}^r$$

The following computations are done

$$K_{b_r}^r = K_{bb}^r - K_{bi}^r \cdot K_{ii}^{-1^r} \cdot K_{ib_r}$$

$$P_b^{*r} = K_{bi}^r \cdot K_{ii}^{-1^r} \cdot P_i^r$$

$$K_{eff} = \sum_r K_b^r$$

$$P_{eff} = -\sum_r \overline{P}_b^{*r} + P_b$$

Structure Level Computation. Equations of equilibrium for complete structure are solved to get response of the structure. For static equilibrium, we have

$$\mathbf{KU} = \mathbf{P}$$

where

$$\mathbf{K} = \sum_n \mathbf{K}_e, \qquad \mathbf{P} = \sum_n \mathbf{P}_e$$

For dynamic response

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{KU} = \mathbf{P}$$

where

$$\mathbf{M} = \sum \mathbf{M}_e, \qquad \mathbf{C} = \sum \mathbf{C}_e, \qquad \mathbf{P}_e = \sum \mathbf{P}$$

For nonlinear analysis

$$(K_L^t + K_{NL}^t)\ \Delta U = P^{t+\Delta t} - F^{t+\Delta t}$$

For buckling analysis

$$(\mathbf{K} + \lambda \mathbf{K}_\sigma)\ \mathbf{U} = 0$$

For frequency analysis

$$(\mathbf{K} + \lambda \mathbf{M})\ \mathbf{U} = 0$$

Recovery of Element Level Response. After the structure level response are computed, element level displacements, stresses, strains, and forces are computed

$$\mathbf{u}_e = \mathbf{TU}$$

$$\varepsilon_e = \mathbf{B}\mathbf{u}_e$$

$$\sigma_e = \mathbf{BD}\mathbf{u}_e$$

$$\mathbf{F}_e = \overline{\mathbf{A}}\sigma_e$$

2.3.2 Optimal Structural Design

Optimal structural design is carried out using well-known methods given in Haug and Arora (1979) and Arora and Govil (1977). Important steps of optimal design consist of formulation of cost and constraint function, checking for constraint violations, design sensitivity analysis, design change computations and convergence checks. These steps are listed below.

Problem Formulation. Optimal structural design problem formulation is made using a set of state and design variables. The objective is to minimize the cost function

$$\psi_0(\mathbf{z}, \zeta, \mathbf{b})$$

Subjected to state equation

$$h(\mathbf{z},\mathbf{b}) = 0$$
$$\mathbf{K}(b)\mathbf{y} = \zeta\mathbf{M}(b)\mathbf{y}$$

and constraints

$$\boldsymbol{\psi}(\mathbf{z},\zeta,\mathbf{b}) \leq 0$$

Constraint Checks. Violated displacements, stress, frequency and other constraints are identified. For displacement constraints

$$\psi_i \equiv z_j - z_j^a \leq 0$$

For eigenvalue constraint

$$\psi_i \equiv \zeta_0 - \zeta \leq 0$$

For stress constraints

$$\psi_i \equiv \sigma_{ij} - \sigma_a \leq 0$$

For buckling constraint

$$\psi_i \equiv \sigma_{ij} - \sigma_b \leq 0$$

For design variable constraints

$$\psi_i \equiv b_i - b_{iu} \leq 0, \text{ or } b_{i\ell} - b_i \leq 0$$

Design Sensitivity Analysis. Design sensitivity analysis is done to determine the effect of the change in design δb in b^0. Gradients of function ψ is

$$\frac{d\psi_i}{d\mathbf{b}} = \frac{\partial\psi_i}{\partial\mathbf{b}} + \frac{\partial\psi_i}{\partial\mathbf{z}} \cdot \frac{d\mathbf{z}}{d\mathbf{b}} + \frac{\partial\psi_i}{\partial\zeta} \cdot \frac{d\zeta}{d\mathbf{b}}$$

The following computations are needed to calculate gradients,

1. Linear systems

$$\mathbf{K}\frac{d\mathbf{z}}{d\mathbf{b}} = -\frac{\partial \mathbf{h}}{\partial \mathbf{b}} \qquad \text{for direct differentiation method}$$

$$\mathbf{K}^T \lambda_i = \frac{\partial \psi_i}{\partial \mathbf{z}} \qquad \text{for adjoint variable method}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{b}} = \frac{\partial}{\partial \mathbf{b}} \left(\mathbf{K}(\underline{b})\tilde{\mathbf{z}} - \mathbf{F}(\mathbf{b})\right)$$

2. Nonlinear systems (Ryu and Arora, 1984)

$$\mathbf{K}^T \lambda_i^j = \frac{\partial \psi_i^j}{\partial \mathbf{z}} - \left[\frac{\partial}{\partial \mathbf{z}} (\mathbf{K}\tilde{\mathbf{z}})z\right]^T \lambda_i^j$$

3. Sensitivity analysis of eigenvalues

$$\frac{\partial \zeta_j}{\partial \mathbf{b}} = \mathbf{y}_j^T \left(\frac{\partial \mathbf{K}}{\partial \mathbf{b}} - \zeta \frac{\partial \mathbf{M}}{\partial \mathbf{b}}\right) \mathbf{y}$$

Design Change Computation. A change in design variable vector b is computed to reduce the cost. Mathematical programming methods such as the gradient projection or other methods (Arora, et al., 1984b) are used to compute design change δb.

$$b^{\nu+1} = b^{\nu} + \delta b^{\nu} \qquad \nu = 0,1,2,\ldots\ldots\text{iterations}$$

A general flow diagram for optimal design of the structure is given in Fig. 2.3.1.

2.4 Components of a Computer-Aided Structural Design System

Computer-aided structural design system consists of three important components -- a database, a program library, and a communication subsystem. In this study, database which is the most important component of the system is considered in detail. A database contains data required for finite element analysis and structural design optimization. Several users can operate on the database either interactively or though application programs. Thus, a database acts as a central repository of data for CAD applications. The second component, namely, a program library contains both the modules used for data management and modules containing algorithms needed for structural analysis and design applications (matrix operation library, equation solvers, finite element programs and optimization routines). Data management programs have components -- file management, input-output processor, memory management, addressing and searching, and security and protection routines. Finally, the communication subsystem provide link between computer and designer. They also provide channels of data communication between the database, database management, and application programs. A communication subsystem consists of interactive command processors, data definition language, data manipulation language, and routines for graphic display. The basic components of a computer-aided structural design system are schematically shown in Fig. 2.4.1.

Thus, we need to design and develop these three components to provide an efficient and economic means of designing a structural systems using computer.

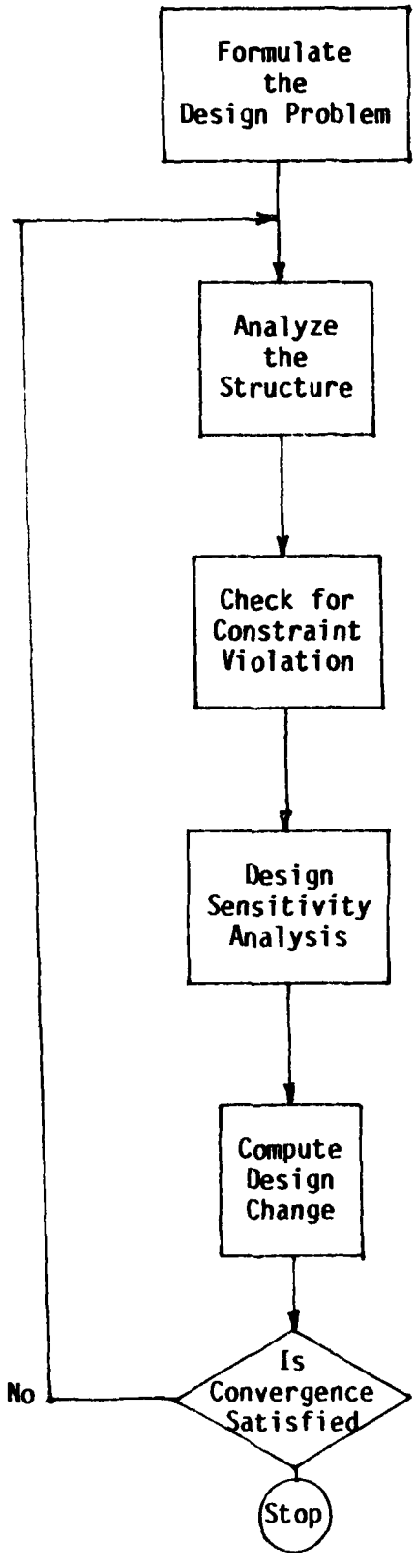


Figure 2.3.1 A General Flow Diagram for Optimal Design of Structure

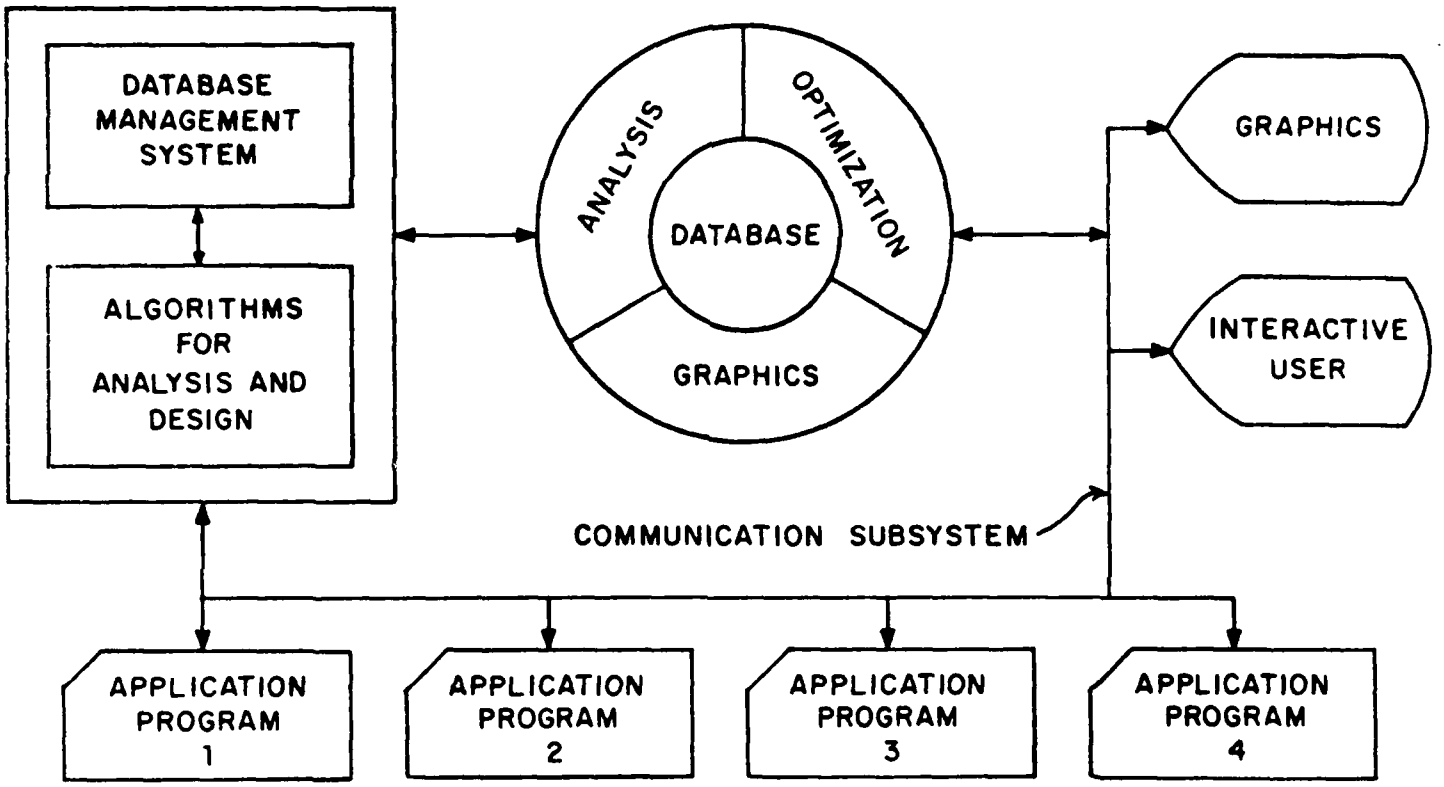


Figure 2.4.1 Components of Computer-Aided Structural Design System

2.5 Database for Computer-Aided Structural Design

We have described nature of the structural design process including finite element analysis and optimization. Also, we observed that large amount of computation is needed in the design process. Finite element analysis and optimization programs generate large amount of data depending on the size of the problem. The amount of data used during design optimization stage depends directly on the number of iterations. Moreover, several application programs are used during the design process and each of them requiring specific data. Therefore, a careful consideration of data organization in a database is necessary to improve design efficiency.

Need for a database in structural design optimization is more important and demanding than that for structural analysis. It is important to realize that design optimization and analysis are fundamentally different in nature. In analysis, generally solution exists and algorithms use data only a few times during solution procedure. In optimal design, existence of even a nominal design satisfing constraints is not assured. Several algorithms may be needed which use similar data to arrive at optimal design. In such a case data used by one algorithm should be made available for use in another algorithm. Therefore, designer needs control to select appropriate algorithm and data to obtain optimum solution. Another important feature of design database is that it contains both informative data such as geometry, material property as well as operational data such as stiffness matrix. Informative data remains static where as operational data gets continuously updated, modified and deleted. A centralized database is needed which stores all the data of analysis and design. A centralized database provides an option fo the designer to interrupt the program execution and provides flexibility for the designer to change the design parameters.

The question of how the database has to be organized?; what kind of information is to be stored?; what kind of database management system (DBMS) is suitable?; how data is manipulated?; how various applications use the data?, have to be studied in detail. As new methods of design evolve, there is need to incorporate the information required for them in the database. Thus it is necessary for the existing database to be flexible and allow simple modifications. The increasing size of database and complexity of information content introduces a new dimension to the problem of in consistency of data. The operational data creates update consistency problems. If informative data are changed, the operational data must be invalidated. Thus, the problem of data dependency which arises from storing operational data together with informative data influence the design of database.

Abstract structural design information must somehow be modeled into the computer. This modelling aspect of actual design data requires a formal approach. In this regard, there exist some guidelines that have been developed in commercial database management area. But the question arises as to whether engineering data could be similarly modeled. If so, do the structural design databases require different performance consideration from the database of commercial applications? These questions have not yet been adequately answered. Several different approaches have to be taken; for example, use of commercially available database systems, and development of special structural design database systems. Further, the data modelling considerations depends on the type of user. Users in structural design can be grouped into system program developers, application programmers, and

interactive users. Requirements of these users have to be considered while designing a database system.

In summary, we have identified the need for a database for structural design and special nature of the data was highlighted. Problems associated in providing a good database were posed. All these aspects must be considered for providing a good database organization for structural design.

2.6 Communication Subsystem

In this section, we emphasize the need for a good communication subsystem of computer-aided structural design system. A good communication link is possible through a well defined languages for interaction between the computer and the designer. Also, computer graphics provides an effective channel of communication.

Languages for interaction are used either through application program or interactively using a computer terminal. Application programs interact with the computer to define and manipulate data in the database. Data definition and data manipulation languages are provided for this purpose. These languages are generally a set of commands in the host language. It is essential to design these languages such that they are simple and easy to use. Query language is used to define and manipulate data interactively. A general set of interactive commands must be available in the system. Requirement and implementation of these languages are discussed in later chapters.

Finite element analysis and design optimization algorithms produce huge amount of results. In order to make these results useful for interpretation and evaluation, they need to be presented in a readily understandable form. Long list of printed data are not suited for comprehension. Graphical presentations are appropriate solutions. Typical tasks of computer graphics include the selection of visual aids (graphic displays, fast plotters), editing of data to be displayed, command interpretation and graphic database management.

Therefore, a well-developed command language and computer-graphics can offer considerable aid to designer to communicate effectively with the computer during the design process.

2.7 Users of Computer-Aided Structural Design System

We have to identify different types of users and their needs in using computer-aided structural design system. Three types of users are identified -- (i) the system programmers (ii) application programmers and (iii) interactive users. The difference between these users and the way they use the system are described below.

System programmers are those who develop general purpose programs for structural analysis and optimization. In general, persons who write these programs are not the same as those who apply them. They work on a very high level of data abstraction. They need a good database management system,

matrix operation and utility library, and a graphic system. There exists a second category of system programs who modify the existing general purpose finite element programs to add new capability to the programs. Some programs like GIFTS, ADINA, and SPAR have excellent analysis capability. However, many of these systems do not have database management capability that is capable of sharing data outside the program environment. In some of these programs, a local data management routines are used. These programs can be incorporated into structural design system by providing an interface between the database and the programs or by modification of program to retrieve essential data that program requires. Pre- and post processing capabilities of these programs together with their local database may be used to integrate them in the design process.

The application programmers are much closer to practical applications. Their interest is not to provide means for general problem solution, but to solve special problems; e.g., stress analysis of a structure for a certain number of load combinations. In general the packaged programs may not have capabilities to handle special needs of the problem. For this reason the application programmers need capabilities to exchange data between subsystems and add their own algorithm whenever the subsystems are not completely covering their needs. Consider, for example a finite element package in which capability to include special boundary condition do not exist. Application programmer in that case selects an appropriate algorithm for assembly and solution of system of equations to meet the needs. Design optimization procedures have similar needs for selecting alternate application programs. Depending on convergence and other requirements, designer switches to appropriate optimization algorithm, but essentially using almost the same data.

Interactive users are those who use the same application program many times by changing only certain parameters. This type of user does not worry about complicated descriptive or algorithmic facilities. Their concern is data input for many iterations with minimum effort on their side and easy-to-perceive representation of output. An interactive user of finite element system may like to see the effect of introducing a boundary condition at a particular node or a load at a node. A designer using an optimization package may like to see the convergence pattern for various values of step size increment in the minimization path.

Thus, we have identified the various types of users of a computer-aided structural design system. This is schematically shown in Fig. 2.7.1.

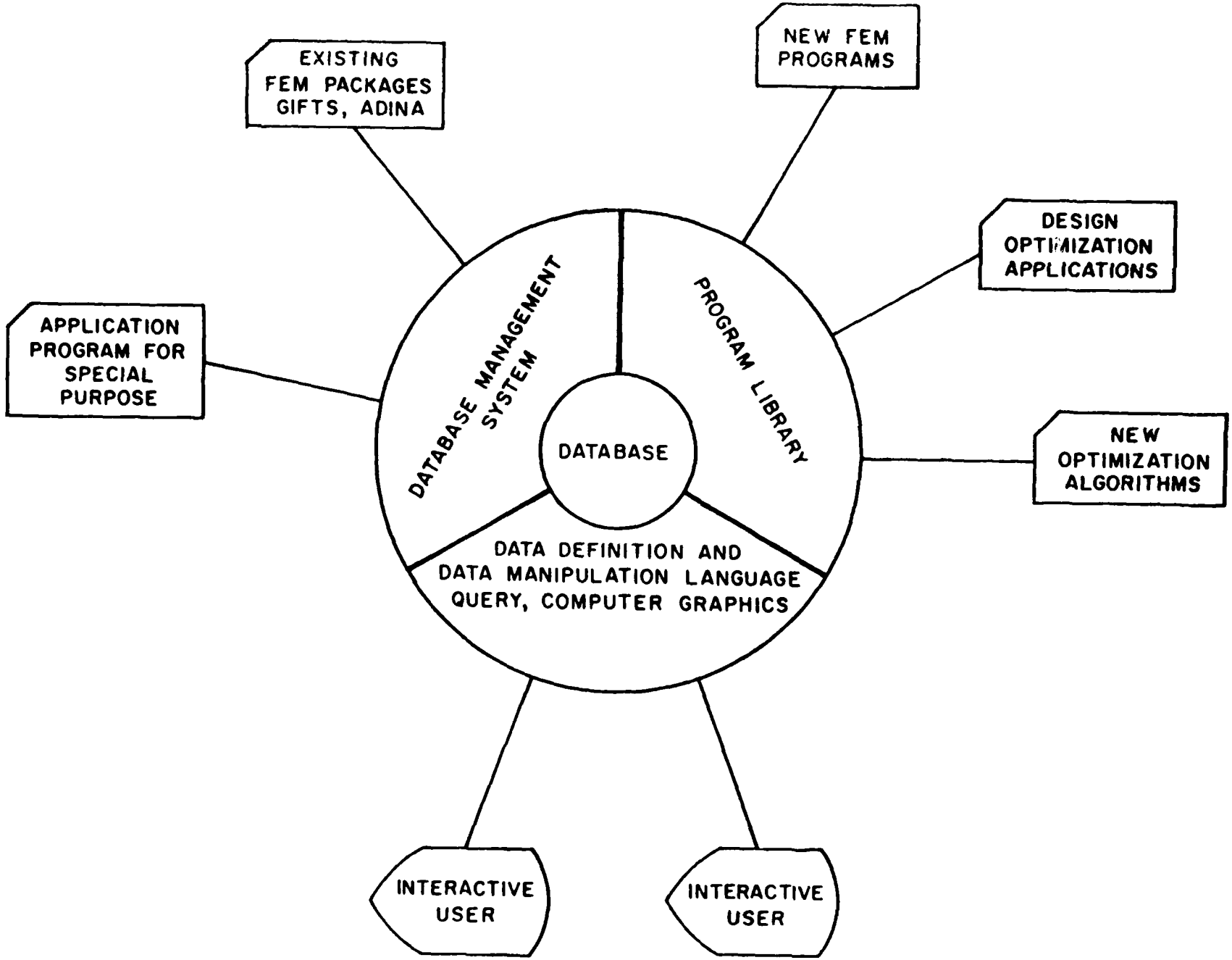


Figure 2.7.1 Users of Computer-Aided Structural Design System

3. DATABASE MANAGEMENT CONCEPTS

3.1 Introductory Remarks

In the previous chapter, we studied the structural design process and emphasized the importance of database management concepts in computer-aided structural design. Now, our problem is how to organize data in a database, what kind of information is to be stored, what kind of database management system is suitable, and how data is manipulated and used. If function of a database were to merely store data, its organization would be simple. Most of the complexities arise from the fact that it must also show association between the various stored data. In this regard, sophisticated techniques are available in business data management area to deal with complex data organization problems. However, techniques used in existing finite element programs are primitive and difficult to use. Therefore, a study of database management concepts is made to understand various methods available for data organization and to implement them for structural design applications. In this chapter, the concepts are explained with reference to finite element analysis and design optimization examples.

In Section 3.2, various database management terminologies are described, since they are relatively new to engineering community. In Section 3.3, commonly used data models are described. Concepts of normalization of data is given in Section 3.4. Semantic integrity and consistency, and transaction management concepts are explained in subsequent sections. Finally, the concept of global and local databases is explained in Section 3.7.

3.2 Definition of Various Terminologies

A number of terminologies and definitions are given to facilitate descriptions in subsequent chapters. They start with simple ones and move on to more complex ones.

Database. A database is defined as a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications. The data are stored so that they are independent of programs which use them. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within the database. The data is structured so as to provide a foundation for future application development. One system is said to contain a collection of databases if they are entirely separate in structure (Martin, 1977).

Logical Data Structure. Data in a particular problem consists of a set of elementary items of data. An item usually consists of single element such as integer, real and character or a set of such items. The possible ways in which the data items are structured define different logical data structures. Therefore, it is the data structure as seen by the user of the DBMS without any regard to storage details.

Model. The logical structure of data

Schema. The coded form of logical data structure is called schema.

Data Independence. It is the property of being able to change the overall logical or physical structure of data without changing the application program's view of the data (Martin 1977).

Entity. An entity may be 'anything having reality and distinctness of being in fact or in thought' (Vetter and Maddison, 1981). An entity may be: (i) a real object like structure, material; (ii) an abstract concept like finite element, nodes, a time period; (iii) an event, i.e., a situation that something is happening (e.g., vibrating structure); and (iv) a relationship, e.g., elements of a particular type.

Entity Set. An entity set is a collection of entities of the same type that exist at a particular instant, e.g., set of finite elements (ELEMENTS) and set of nodes (NODES).

Property. Property is a named characteristic of an entity, e.g., element name, and element material type. Properties allow one to identify, characterize, classify and relate entities.

Property Value. It is an occurrence of a property of an entity, e.g., 'element name' has property value BEAM.

Entity Type. Entities having same kind of properties are said to be of same type. Upper case letter is used for its name.

Domain. A domain is the set of eligible values for a property. A domain has same characteristics as a set, i.e., the values belonging to a domain are distinct and their order is immaterial. A predicate is associated with each domain allowing one to determine whether a given value belongs to the domain in question. Thus the formal definition of domain D_i is

$D_i = \{v_i | P_i\}_{p_i}$ where v_i represents a value satisfying the predicate

Examples of domain are

Element Name = {BEAM, TRUSS, ...}
Element Material type = {STEEL, ALUMINUM, ...}
Length = $\{x | x > 0 \text{ and } x < 100\}$

Attributes. Columns of a two-dimensional table are referred to as attributes. An attribute represents use of a domain within a relation. Attribute names are distinct from those of the underlying domains; e.g.,

Domains:	NODES = $\{i \mid i > 0 \text{ and } i < n\}$
	DOFS = $\{j \mid j > 0 \text{ and } j < m\}$
Attributes:	NODE1 - First node of an element derived from domain NODES
	DOF1 - First d.o.f. derived from domain DOFS
Relation:	ELEMENT(E#, NODE1, NODE2)
	Attributes NODE1, NODE2 and derived from domain NODES

Entity Key. An entity key is an attribute having different values for each occurring entity and provide unique identification of a tuple. An entity represents a compound key if it corresponds to a group of attributes. It is also called candidate key.

Primary Key. If several entity keys exist for a given entity set, then one of them is arbitrarily choosen as the primary key.

Secondary Key. It is an attribute that does not have different values for each occurring entity, but identifies those occurring entities that have certain property.

Relation. R is a relation on the domains (i.e., sets) D_1, D_2 $\cdots$ D_n (not necessarily distinct) if it is a subset of cartesian product $D_1 \times D_2 \times \cdots \times D_n$. Thus $R \subseteq D_1 \times D_2 \times \cdots \times D_n$. The value n represents the degree of the relation R. The relation R is usually written as

$$R(D_1, D_2, \cdots D_n)$$

Here D_1, D_2 $\cdots$ D_n are called attributes of the relations. The values of the attributes are taken from the corresponding domains for D_1, D_2, $\cdots$ D_n. Note that domain is a set of values where as attribute is a list of values (Vetter and Maddison, 1981).

Tuple. Rows of a relations are called tuples.

Function. A function is a special kind of relation between two sets, say, A and B. Each member of a set A is associated with exactly one member of set B. A function f is denoted as

$$f: A \rightarrow B$$

Functional Dependence. An attribute A is functionally dependent on the attribute B of a relation R if at every occurrence of B-value is associated with no more than one A-value. This is denoted as

$$R.B \rightarrow R.A$$

Example. As an example, consider the relation
ELEMENT (ELMT#, EL-NAME, AREA)
EL-NAME is functionally dependent on ELMT#. AREA is functionally dependent on ELMT#. ELMT# is not functionally dependent on EL-NAME, because more than one element could have the same name. Similarly, ELMT# is not functionally dependent on AREA.

An attribute can be functionally dependent on a group of attributes rather than one attribute. For example, consider the relation for a triangular finite element:
CONNECTION (NODE1#, NODE2#, NODE3#, ELMT#)

Here ELMT# is functionally dependent on three nodes NODE1#, NODE2#, and NODE3#. Given any one of NODE1#, NODE2#, or NODE3# it is not possible to identify ELMT#. These functional dependencies are shown in Fig. 3.2.1.

Full Functional Dependency. An attribute or a collection of attributes A of a relation R is said to be fully functionally dependent on another collection of attributes B of R if A is functionally dependent on the whole of B but not on any subset of B. This is written as

$R.B \rightarrow R.A$

In the Fig. 3.2.2 for example, ELMT# in the relation CONNECTION of a triangular finite element is fully functionally dependent on concatenated attributes NODE1#, NODE2# and NODE3# because three nodes combined together define an element. NODE1#, NODE2#, or NODE3# alone does not identify ELMT#.

Transitive Dependence. Suppose A, B and C are three distinct attributes or attribute collections of a relation R. Suppose the following dependencies always hold: C is functionally dependent on B and B is functionally dependent on A. Then C is functionally dependent on A. If the inverse mapping is nonsimple (i.e., if A is not functionally dependent on B or B is not functionally dependent on C), then C is said to be transitively dependent on A (refer to Fig. 3.2.3). This is written as

$R.A \rightarrow R.B$
$R.B \not\rightarrow R.A$
$R.B \rightarrow R.C$

Then, we can deduce that

$R.A \rightarrow R.C$
$R.C \not\rightarrow R.A$

For example, consider the relation
EL-DISP(ELMT#, EL-TYPE, DOF/NODE)

Here, ELMT# → EL-TYPE
EL-TYPE ↛ ELMT#
EL-TYPE → DOF/NODE

Therefore ELMT# → DOF/NODE (transitively dependent)
DOF/NODE ↛ ELMT#

Digraph. A directed graph (refer to Fig. 3.2.4) or a digraph is a figure with nodes and arcs. Each arc is a line with a direction. Nodes represents attributes and arcs represent dependencies (functional, fully functional). Length of a path is the number of arcs in it. For example N_1-A_1-N_2-A_3-N_3-A_4-N_2-A_5-N_4 has length 4. Distance between two nodes is the longest possible path between them. For example distance between N_1 and N_4 is 4 since longest possible path between nodes is N_1-A_1-N_2-A_3-N_3-A_4-N_2-A_5-N_4 (Vetter and Maddison, 1981).

Connectivity Matrix. Square matrices can be used to represent digraphs. If the digraph has n nodes then an nxn square matrix is used. Rows and columns represents nodes. Using the value 1 to means presence of a connection and the value 0 to mean absence of a connection, a square matrix can represent the connection between the nodes of a digraph. For example, if node 2 is connected to node 4, then connectivity matrix is

N_1	N_2	N_3	N_4
0	0	0	0
0	0	0	1
0	0	0	0
0	0	0	0

Figure 3.2.1 Functional Dependencies

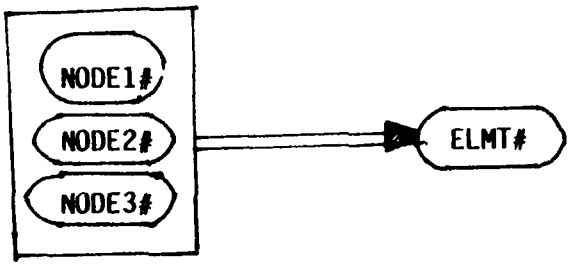


Figure 3.2.2 Full Functional Dependency

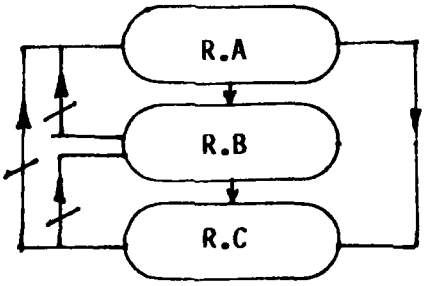


Figure 3.2.3 Transitive Dependence

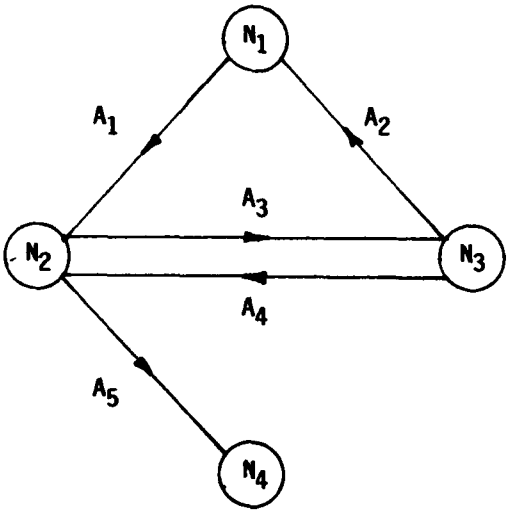


Figure 3.2.4 Digraph

Each row and column of a connectivity matrix is named the same as the corresponding node (i.e., with the name of the attribute constituting the node). Appropriate role names are then used to distinguish multiple rows and columns denoting essentially a single node (Vetter and Maddison, 1981).

3.3 Views of Data and Data Models

Formal description of data and associations among them are essential for good data organization. The most elemental piece of data is a data item. It cannot be further subdivided into smaller data type. A data item by itself is not of much use. It becomes useful only when it is associated with other data items. Thus, database consists of data items and the association among them.

STRUCTURAL MEMBER MATERIAL

A data model is nothing but a map showing different data items and their associations. A data model shows logical organization of data and is useful to describe user's view of data. Layout of data on physical storage devices is known as physical organization.

A database can be viewed at various levels depending on the context. A level of data that represents view of interactive terminal users and application programmers are known as external view. Conceptual view of data deals with inherent nature of data occurring in real world information and represent global view of data. The data organization that describe the physical data layout is dealt at internal level. At this level, one is concerned with efficiency and storage space details. There is one more level of data organization below the internal level where the actual storage of data on a particular computer system becomes the main consideration. But, this aspect is a specialists job and has no general guidelines. Therefore, it is not discussed here. Three levels of data - external, conceptual and internal are used to describe various views of data. These levels of data organization were suggested by ANSI/SPARC (Standard Planning and Requirements committee). It is gaining wide acceptance in designing a database.

The data associations at various levels mentioned above are described by three common approaches - viz hierarchical, network and relational. These are described in detail in the following subsections.

3.3.1 Hierarchical Model

In this model, the data is represented by a simple tree structure. A tree is composed of a hierarchy of elements called nodes. Every node has a node related to it at a higher level. The node at a higher level is called a parent node. Each node can have one or more nodes related to it at a lower level called child nodes. A node at the top of a tree is called the root. No node can have more than one parent. A hierarchical model has one-to-many relationships. In finite element analysis, we can form a hierarchical model with data items such as structure, substructure, elements and nodes. This model is schematically shown in Figs. 3.3.1 and 3.3.2.

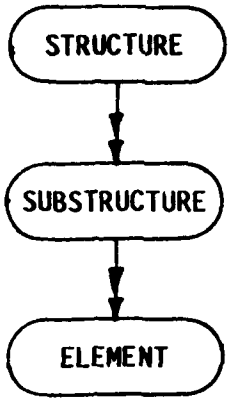


Figure 3.3.1. Hierarchical Data Model

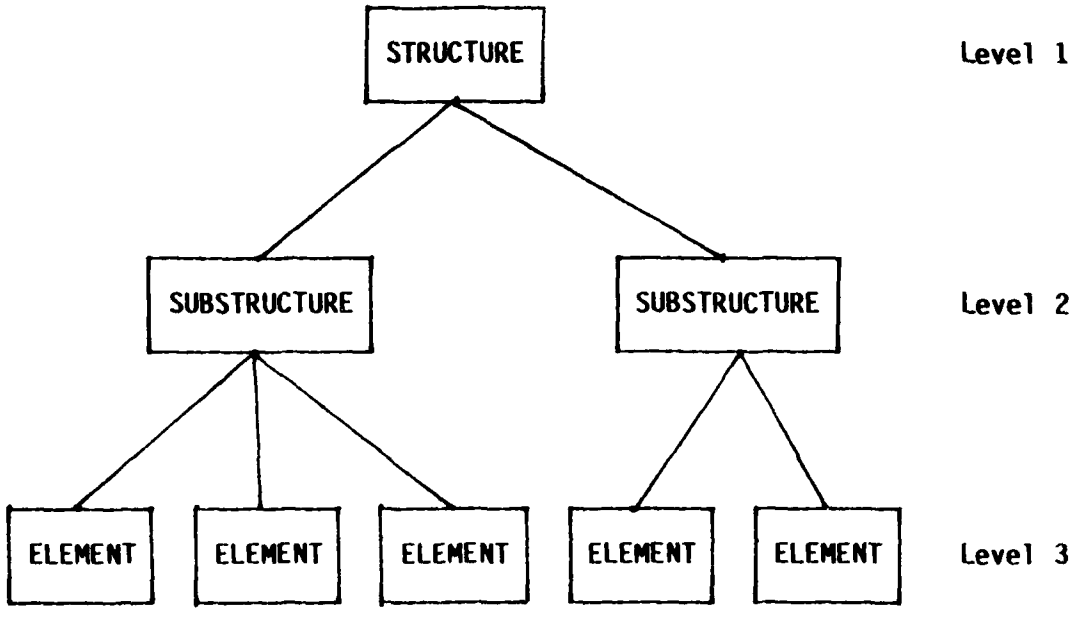


Figure 3.3.2 An Occurrence of a Hierarchical Data Model

3.3.2 Network Data Model

In network data model a child in a data relationship has more than one parent. A network is more general than a hierarchy because a given node occurrence may have any number of immediate superiors as well as any number of immediate dependents. The network model allows many-to-many relationships. A network structure is said to be simple if each directed logical association is functional in at least one direction. This means the model has no line which has double arrows in both direction. Complex network will have double arrows in both directions. Examples of network model for a finite element and node relation is shown in Figs. 3.3.3 and 3.3.4.

3.3.3 Relational Model

Given a collection of sets D_1, D_2 $\cdots$ D_n (not necessarily distinct), R is a relation defined on the n sets if it is a set of ordered n-tuples $<d_1, d_2 \cdots d_n>$ such that d_1 belong to D_1, d_2 belong to D_2, $\cdots$, d_n belong to D_n. Sets D_1, D_2 $\cdots$ are the domains of R. The value n is the degree of R. A data model constructed using relations is referred to as a relational model. A relational data model has a tabular form of data representation. Figure 3.3.5 represent a relational model of data. The rows of the table are referred to as tuples. The columns are referred to as attributes. Relations of degree one are said to be unary. Similarly relations of degree two are binary, relations of degree three are ternary, and relations of degree n are n-ary. The relational model provide an easy way to represent data and is simpler to use than hierarchical or network data model. The relations can be easily manipulated using special relational operators such as PROJECT, JOIN, etc. The operator PROJECT yields a 'vertical' subset of a given relation, i.e., subset obtained by selecting specified attributes. The operator JOIN puts together columns from different relations. An example of relational model in finite element analysis is node-coordinate relation shown in Fig. 3.3.5.

3.3.4 Advantages and Disadvantages of Data Models

It was seen that various types of data models described in the previous sections could be used for structural analysis and design optimization applications. However, it is not possible to expect effective use of any one of the models in all situations. Hierarchical data model is clearly superior in organizing general engineering data which occur naturally in hierarchical form. For example, large order matrices can be assembled using submatrices. These submatrices can be organized at various hierarchical levels. Network model handles more general situation than hierarchical model. The disadvantage of this model is the complexity associated with its use. Both hierarchical and network data model use a fixed structure and offer little flexibility to change for alternative structure. However, a relational data model uses a less preconceived structure and provides user-firendly data representation. This model can provide easy access to data for the user. Also, the tabular structure of the model provide a convenient way of representing structural design data that are a convenient way of representing structural design data that are generally in the tabular form. It is possible

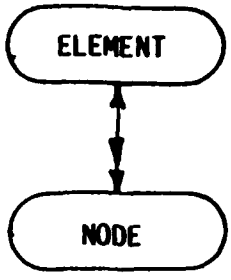


Figure 3.3.3 A Network Model

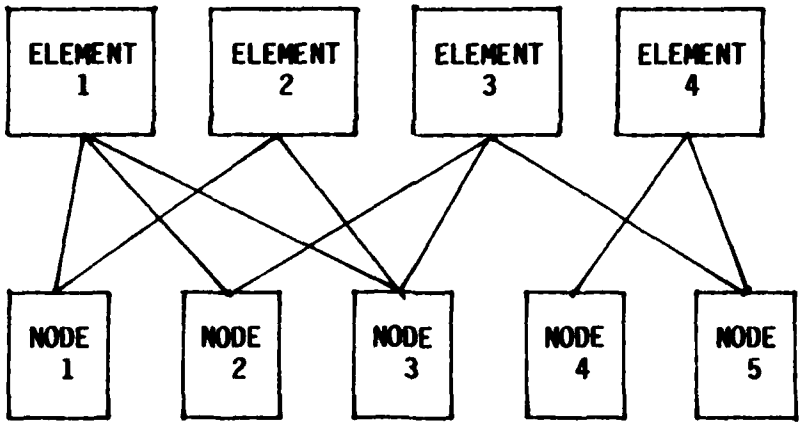


Figure 3.3.4 An Occurrence of a Network Model

CORD

NODE NO	X	Y	Z

Figure 3.3.5 Relational Model

to support simple query structure in the relational model. In situations where application programs require a complete set of related items together, retrieving parts of information is not useful. In such a case the relational model which is set oriented provides a suitable way to organize the design data.

Relational data model is therefore selected for design of the database and the development of database management system for structural analysis and optimization applications.

3.4 Normalization of Data

It was seen in the previous section that data items are grouped together to form associations. An issue of concern here, is how to decide what data items have to be grouped together? In particular, using a relational model, determining what relations are needed and what their attributes should be? It was emphasized in Chapter 2 that structural design data gets constantly modified, updated and deleted. As database is changed, older views of data must be preserved so as to avoid having to rewrite the programs using the data. However, certain changes in data associations could force modification of programs, and could be extremely disruptive. If grouping of data items and keys is well thought out originally, such disruptions are less likely to occur.

Normalization theory (Date 1982) provides certain guidlines to organize data items together to form relations. The theory is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. Three normal forms - 1^{st}, 2^{nd} and 3^{rd} - are described below.

First Normal Form (1NF). A relation is said to be in first normal form if and only if it satisfies the constraint of having atomic values.

As an example, Fig. 3.4.1 shows the relation CONN between four attributes ELMT#, E-NAME, NODES#, and DOF/NODE with domains D_1, D_2, D_3 and D_4. The relation is first shown not in 1NF and then it is shown in the 1NF.

Second Normal Form (2NF). A relation is in second normal form if and only if it is in 1NF and every non-key attribute is fully functionally dependent on each candidate key.

Let us see if the relation CONN of Fig. 3.4.1 in the 1NF is also in 2NF. Consider a non-key attribute E-NAME:

ELMT#, NODES# $\rightarrow$ E-NAME
ELMT# $\rightarrow$ E-NAME
NODE# $\not\rightarrow$ E-NAME

Therefore, ELMT#, NODES# $\not\Rightarrow$ E-NAME, i.e., E-NAME is not fully functionally dependent on (ELMT#, NODES#).

Domain D_1	Domain D_2	Domain D_3	Domain D_4
ELMT#1	BEAM	NODE#1	No. of DOF per node
ELMT#2	TRUSS	NODE#2	6
ELMT#3	PLATE	NODE#3	3
		NODE#4	2
		NODE#5	

CONN

Key ELMT#	E-NAME	Key NODES #	DOF/NODE
1	BEAM	1 2	6
2	TRUSS	3 5	3
3	PLATE	2 3 4 5	2

Not in 1NF

Key ELMT#	E-NAME	Key NODES #	DOF/NODE
1	BEAM	1	6
1	BEAM	2	6
2	TRUSS	3	3
2	TRUSS	5	3
3	PLATE	2	2
3	PLATE	3	2
3	PLATE	4	2
3	PLATE	5	2

In 1NF

Figure 3.4.1 First Normal Form for a Relation CONN

Similarly for the non-key attribute DOF/NODE:

ELMT#, NODES# → DOF/NODE
ELMT# → DOF/NODE
NODE# ↛ DOF/NODE

Therefore, ELMT#, NODES# ⇏ DOF/NODE. Since neither E-NAME nor DOF/NODE are fully functionally dependent on candidate key (ELMT#, NODES#) the relation CONN is not in 2NF.

Conversion of the relation CONN to 2NF consist of replacing CONN by two of its projections (refer to Fig. 3.4.2).

NAM-DOF ← CONN (ELMT#, E-NAME, NODES#, DOF/NODE)
ELMT-NODE ← CONN (ELMT#, E-NAME, NODES#, DOF/NODE)

Relation ELMT-NODE does not violate 2NF because its attributes are all keys.

Third Normal Form (3NF). A relation is in third normal form if it is in second normal form and every non-prime attribute of relation is non-transitively dependent on each candidate key of the relation.

For example, consider the relation NAM-DOF (Fig. 3.4.2) to see if it is in third normal form. It still suffers from a lack of mutual independence among its non-key attributes. The dependency of DOF/NODE on ELMT#, though it is functional, is transitive (via E-NAME). Each ELMT# value determines an E-NAME value and in turn determines the DOF/NODE value. This relation is reduced further into relations NAME and DOF. These relations (Fig. 3.4.3) are in third normal form.

3.5 *Semantic Integrity and Consistency*

Semantic Integrity of a database has specific meaning. It refers to the correctness of the values in the database. The problem of integrity is to ensure accuracy of data in the database. The structural design database is highly volatile. Designers continuously build up the database and assign its contents. During generation of operational information, new data items are added, association among them are formed, and existing definitions may be deleted. Therefore any indicriminante grouping of data items can lead to semantic integrity problems. The problems become more severe if a number of users or application programs use the database without appropriate integrity checks. This is because it is possible for a program with errors in it to generate bad data and thus spoil other innocent programs using that data. Thus characteristics of the design database, such as stated above, make the maintenance of integrity both crucial and difficult.

To enforce integrity of a database, integrity rules (Date, 1982) are imposed. The two common type of intergrity rules are entity integrity and referential integrity. The entity integrity rule specifies that "no component of a primary key may be null". This requirement implies that all entities must be distinguishable which means that they must have a unique identification of some kind. Primary keys provide a means for unique identification in a relation. For example if finite elements are entities, then element number form the primary key and hence must be unique. The second type of integrity rule (referential integrity) deals with the inter-relational aspects of relations. It is common for one relation to include reference to another. Consider for example a relation ELEMENT which has attributes ELMT#,

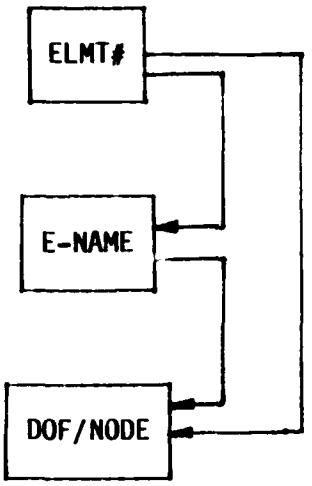
NAM-DOF

ELMT#	E-NAME	DOF/NODE
1	BEAM	6
2	TRUSS	3
3	PLATE	2

ELMT-NODE

ELMT#	NODE#
1	1
1	2
2	3
2	5
3	2
3	3
3	4
3	5

Figure 3.4.2 Second Normal Form for Relation CONN



NAME

ELMT#	E-NAME
1	BEAM
2	TRUSS
3	PLATE

DOF

E-NAME	DOF/NODE
BEAM	6
TRUSS	2
PLATE	2

Figure 3.4.3 Third Normal Form for Relation NAM-DOF

and nodal connectivity NODE1, NODE2 and NODE3, and another relation NODES which has attributes NODE# and coordinates X, Y, and Z. Here, relation ELEMENT includes references to relation NODES via its NODE1, NODE2, and NODE3 attributes. It is clear that if a tuple of ELEMENT contains a value for NODE1 say n, then a tuple for n must exist in NODES (otherwise, the ELEMENT tuple would apparently be referring to a nonexistent node). The referential integrity rule is "Let D be a primary domain, and let R_1 be a relation with an attribute A that is defined on D. Then, at any given time, each value of A in R_1 must be either (a) null, or (b) equal to v, where v is the primary key value of some tuple in some relation R_2 (R_1 and R_2 not necessarily distinct) with primary key defined on D". These rules are enforced in practice by providing key constraints, referential constraints and other constraints. Primary and secondary keys are explicitly specified in database schema. It is possible to enforce other types of constraints. An example is, "coordinates of nodes must lie in the range of -100.0 to +100.0" in relation NODES. Thus, there is no limit to the number of constraints that can be imposed to provide database integrity.

Consistency is a special case of integrity and involves maintaining the equivalence of redundant data. Storing the same data more than once leads to data redundancy. In general, it is not desirable to have data redundancy in database. Data redundancy occurs while catering to needs of different application programs requiring same data in various forms. It also occurs while providing efficient database operations. Suppose that a truss member has area of cross-section A represented in two places in the database - say in ELEMENT relation and DESIGN-VARIABLE relation and that the system is not awae of this duplication. Then there can be some occasion where the two data items will not agree. At such time the database is said to be inconsistent. It is clear that if such redundancy is eliminated then inconsistencies cannot occur. If redundancy cannot be eliminated, then alternate way is to provide a control to ensure changes made to either of the two data items are automatically made to the other. This ensures database consistency.

Integrity and consistency, then are important task in structuring design information. These tasks have to be performed by the computer-aided structural design system which was previously a completely manual operation.

3.6 Transaction Management

In preceding sections, various concepts for data abstraction such as entities, data associations, data models, normalization of data, and intergrity were discussed with reference to structural analysis and design data. It was emphasized that maintenance of integrity of design database is an important task in the environment of highly volatile design database and operations on them by multiple applications. Rules for integrity maintenance discussed in the previous sections are mainly applicable to static information of the data. But the design data also contains operational information which may be classified as dynamic. For example, in finite element analysis, input data such as finite element numbers, nodal connectivity, cross-sectional details, and material property are static information for input module (application program). This static information is used for assembly modules which create operational information such as element stiffness and assembled stiffness matrices. This operational information becomes static for the next

application such as solution module, since they are entry information for it. Information content in a database stabilizes only at the end of analysis and design, until then the database is subjected to successive refinement of schema, values of variables modified, and accumulation of redundant information takes place. This characteristic of the design database makes the maintenance of complete integrity almost impossible. Therefore, schemes have been proposed (Kutay and Eastman, 1983) to provide partial integrity of databases. Conceptual details of transaction management scheme to provide partial integrity of databases are discussed in the following with reference to finite element analysis and structural design optimization.

The transaction concept is the one that is closely related to operation abstraction. This concept suggests that instead of allowing arbitrary operations on a database, these operations must be structured into a set of actions such that when each set is executed, the integrity of the database is maintained. In this way a database has complete integrity during all quiscent periods; loss of integrity are temporary and limited to duration of transaction upon the database. The definition of transaction is given as follows (Kutay and Eastman, 1981):

A database consists of data units called entities. The entities of a database form a distinct set $\{e_1, e_2 \cdots e_n\}$. Each transaction performs its processes on a set of entities called a transaction entity set (TES). A TES has two parts which are not necessarily inclusive: A read set $\{e\}R$ and a write set $\{e\}W$. A transaction can be defined as "A collection of actions on a database that reads entity set $\{e\}R$ and potentially writes into entity set $\{e\}W$. Prior to invokation, it requires that the set of integrity constraints $\{C^+\}^B$ be satisfied on the entities $\{e\}R$ and $\{e\}W$. During the transaction, the integrity constraint set $\{C^-\}^D$ on $\{e\}R$ and $\{e\}W$ may be violated. After successful completion of the transaction, integrity conditions on $\{e\}W$ may have been changed. These are denoted by the sets $\{C^+\}^A$, $\{C^-\}^A$, where $\{C^+\}^A$ are the integrity constraints added by the transaction and $\{C^-\}^A$ are those that are eliminated.

This definition suggest that in a design database a transaction should incorporate entities together with related integrity constraints. The transaction can then be of the form

$$T_i: \quad [\{e\}R, \{e\}W, \{C\}^B, \{C\}^D, \{C\}^A]$$

Such a description is referred to a transaction class.
(Note: $\{e\}R$ entity set read

$\{E\}W$ entity set written

$\{C\}B$ Integrity constraint before transaction

$\{C\}D$ Integrity constraint during transaction

$\{C\}A$ Integrity constraint after transaction

-violated, + satisfied)

The transaction concept is explained with reference to a structural design example. Following transactions are considered:

1. A transaction class T_1 that defines finite element structural idealization by identifving the element numbers, nodal coordinates, and nodal connectivity.

2. A transaction class T_2 that defines forces in individual members.

3. A transaction class T_3 that defines member cross-section details and sizes.

4. A transaction class T_4 that determines the cost of the structural system. Integrity constraints for this system are defined as

   $\{C_1\}$: "Structural idealization defined"

   $\{C_2\}$: "Forces in member defined"

   $\{C_3\}$: "Member sizes defined"

   $\{C_4\}$: "Structural cost estimated"

   Then for this structural system the transaction classes are

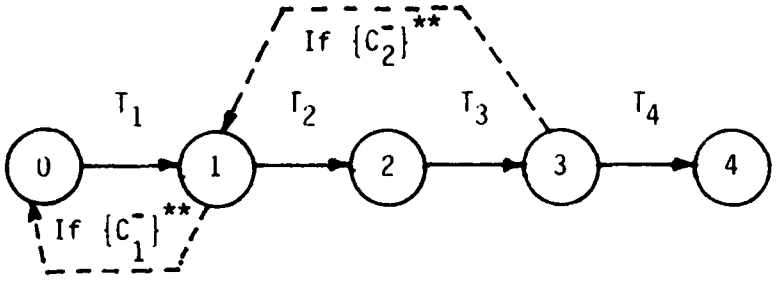
T_1: $[\{\text{Elements, Nodes}\}^{*}_{R}, \{\text{Element, Nodes}\}W, \{C_1^-\}^D, (\{C_1^+\}, \{C_2^-\}, \{C_3^-\}, \{C_4^-\})^A]$

T_2: $[\{\text{Element}\}R, \{\text{Forces}\}W, \{C_1^+\}^B, \{C_2^-\}^D, (\{C_2^+\}, \{C_3^-\}, \{C_4^-\})A]$

T_3: $[\{\text{Elements}\}R, \{\text{Sizes}\}W, (\{C_1^+\}, \{C_2^+\})^B, \{C_3^-\}^D, (\{C_2^-\}^{**}, \{C_3^+\}, \{C_4^-\})^A$

T_4: $[\{\text{elements}\}R, \{\text{cost}\}W, \{C_3^+\}^B, \{C_4^-\}D, \{C_4^+\}^A]$

* indicates that some entities in them are conditional. T_1 to T_4 are marco transactions, i.e., composition of lower level transactions. For example, lower level transactions in T_2 defining forces in elements are transaction for stiffness matrix, displacements, etc. A transaction graph for structural system is given below.



** indicates that they are conditional. If they hold after a transaction is committed, they require that previous transaction should be repeated and this result in iterations. Assume for instance that force in the member is not yet defined. That is, $\{C_2\}$ does not hold for its entities. If T_3 is allowed to execute, it would result in failure of transaction due to insufficient data.

This transaction scheme also supports design iterations in a natural way. Consider, for example, violation of cost minimization at T_4. In this case, an iteration might be necessary to redefine the design variables. The formal definition of transaction class allows representation of iterations in a natural way. To incorporate an iteration of design, a possible violation of $\{C_1\}$ is declared as an after effect of T_4.

These features of a transaction in a design database can be formalized by the following rules about integrity management:

1. If the required $\{C^B\}$ conditions for a transaction class do not hold at any point in time, no transactions of this class are allowed to execute.

2. It is the task of a transaction to satisfy a set of integrity constraints $\{C^+\}^A$ so that other transactions which require them are allowed to execute.

3. A transaction also violates a set of constraints $\{C^-\}^A$ to indicate that other transactions should be invoked to satisfy them.

3.7 Global and Local Databases

Concept of global and local database are important in the light of discussion in Section 2.6 on the nature of application programs. Computer-aided design of complex structural systems uses several application programs during the design process. Many of these programs require common information such as geometry of the structure, finite element idealization details, material properties, loading conditions, structural stiffness, mass and load distributions, and responses resulting from analysis runs. Also, it is common that data generated by one program is required for processing in subsequent programs in certain predetermined pattern. These data do not include transitory information such as intermediate results generated during an analysis run. The transitory information is highly unstructured and its usage pattern is known only to applications that use them. Generally, the transitory information is deleted at the end of a run. Therefore, there is a need for systematic grouping of the data.

A network of databases offers a systematic approach to support data of multiple applications. A network of databases consists of a global database connected to a number of local databases through program data interface. Application programs which use them may be thought as links connecting the databases (Fig. 3.7.1). A global database contains common information required for all applications where as a local database contains only application dependent transitory data. Data in global database is highly structured and integrity of the database is maintained carefully. However,

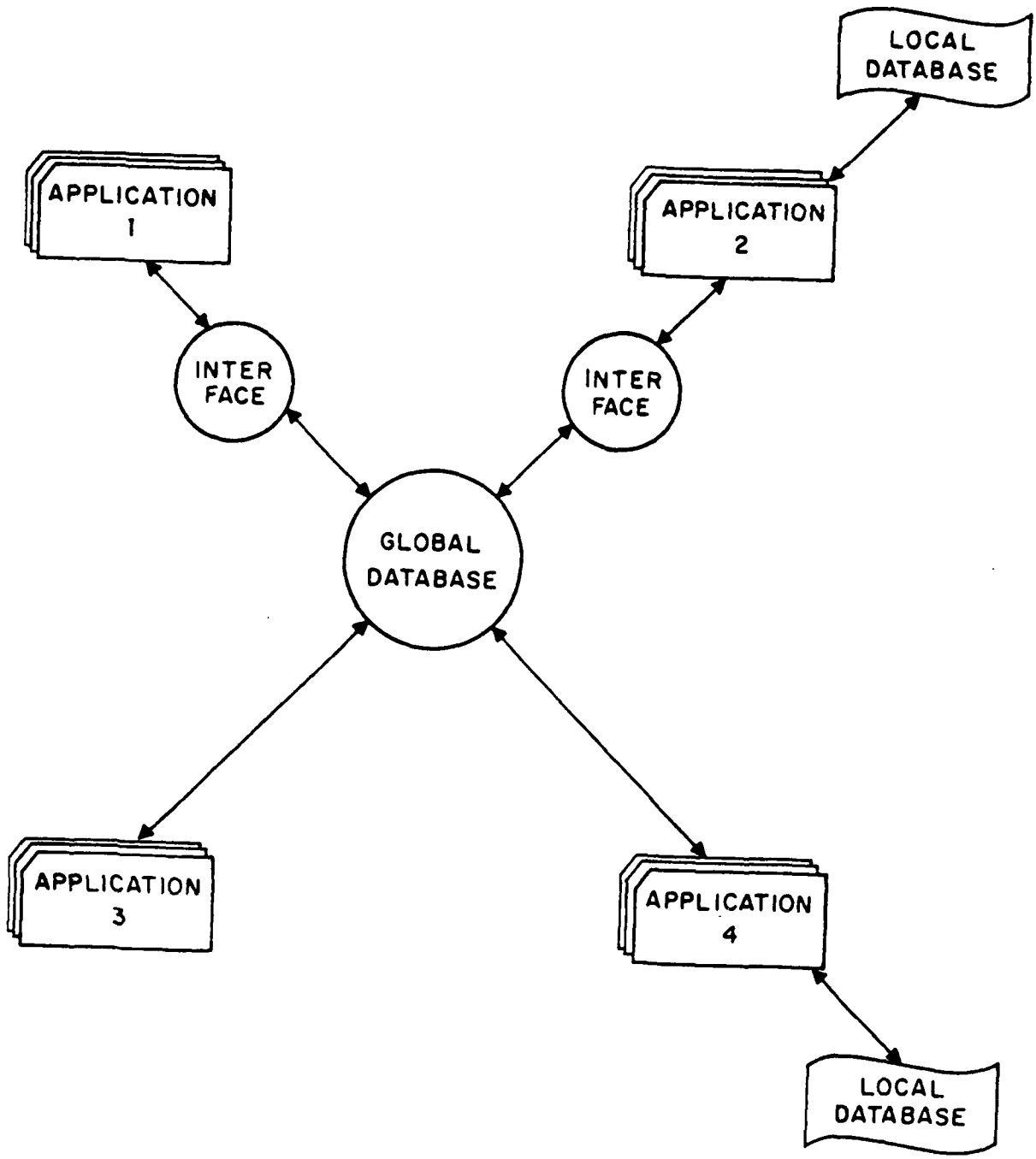


Figure 3.7.1 Network of Databases

data in local database is extremely flexible and integrity is not of importance.

This network of databases offers considerable aid in structural design process. Any changes made to the data in global database is immediately available for use in other applications of the system. Any new application program can be added to share the common data. The data views in global database are clear to all applications and any modified views can be easily incorporated to suit a new application. Local databases are dependent on application programs and are highly efficient in data transfer operations since no overhead is involved in maintaining complicated data structures. It supports trial and error design process by providing scratch pad work space which can be erased from the local database at a specified design stage. Any intermediate results can be stored in a local database. Summarized and final results of design can be transferred to a global database.

4. DATABASE DESIGN METHODOLOGY FOR STRUCTURAL DESIGN

4.1 Introductory Remarks

A methodology for designing databases for structural design is proposed in this chapter. The methodology is based on the database management concepts described in the previous chapter. Till now, no comprehensive study on design of a database for structural analysis and optimization exists in the literature. However, some researchers have discussed some possible ways of data organization for finite element analysis of structures. Lopez (1974) and Pahl (1981) have shown, how an hierarichal data model can be used to organize data of finite element analysis. Relational data model has been recently introduced to structural application and gaining popularity because of its simplicity and ease of use. Fishwick and Blackburn (1982) used relational data model with finite element program SPAR and optimization package PROSSS. But the use of the model was limited to interfacing of these programs using a relational database management system. It is emphasized again that the failure of existing finite element programs like NASTRAN, ANSYS, ADINA, and GIFTS, in providing intermediate results to users for further analysis and design purpose, is due to nonavailability of generalized database management support. out of core data organization in these programs were based on intution, since no scientific database design techniques were available at the time these programs were developed.

In the methodology proposed for database design, relational data model is used. Three levels of data organization -- conceptual, internal and external are suggested for structural design database. In Section 4.2, a methodology for constructing a conceptual data model is described. The conceptual model enables us to find out the inherent nature of structural design data irrespective of computer program constraints. Since large amount of data storage and speed of accessing data is required in finite element analysis and optimization, we need to consider efficiency of storage space and I/O. This aspect is considered in Section 4.3. A methodology for developing an internal model is given there. In Section 4.4, a methodology for developing an external data model is described. An external model can provide data needed for multiple users or application programs according to their individual perspectives of data. Matrix data constitutes a large protion of finite element analysis and optimization data. Such data needs special considerations for accomodation in a database. A methodology for organizing large matrix data is given in Section 4.5. Finally in Section 4.6 algorithmic modelling is given.

4.2 Development of a Conceptual Data Model

4.2.1 Basic Considerations

Capability of a database to support any structural design application depends on the effectiveness of the data model devised for the system. Our objective now is to construct a conceptual data model at a suitable level of abstraction regardless of whether or not the available database management software supports such model directly. The conceptual data model should serve as a central reference point for all users of the database. It changes only if changes in the structural design process occur. Any change in the database

management software of application program should not affect the conceptual data model. The model should be capable of supporting new applications with the existing types of data as well as incorporating further data types as needed. Therefore, an analysis of data used in structural design is made to incorporate the features of data model described above. In the analysis, the information in use or needed in future is identified, classified and documented. This forms a basis for a conceptual data model to represent structural design data and design process as a whole.

The following steps are proposed to develop a conceptual data model. These steps are discussed in detail in Subsections 4.2.2 to 4.2.5.

1. Identify all the characteristics of data used in structural analysis and design optimization.

2. Data identified are stored in a number of relations. Reduce these relations to elementary relation which represent inherent association of data.

3. Derive further set of elementary relations from the elementary relations formed in Step 2. This step will uncover more relationships between basic data collected in Step 2.

4. Remove redundant and meaningless relations obtained in Step 3 to get a conceptual data model.

The conceptual model obtained by this process is abstract, representing inherent nature of structural design data and is independent of any computer restraint or database management software support.

4.2.2 Identification of Characteristics of Data

The following steps are proposed to identify the characteristics of data used in structural design.

Step 1. Identify each type of entity and assign a unique name to it.

Step 2. Determine the domains and assign unique names to them. This step identifies the types of information which will appear in some part of the model, such as attributes.

Step 3. Identify the primary key for each type of entity depending on the meaning and use.

Step 4. Replace each entity set by its primary key domains. Determine and name relations corresponding to association between primary key domain and other domains. This step gives a collection of relations.

Modified definition of domain and attribute are suggested as follows to suit correct identification of structural design information.

Domain. A domain is the set of eligible values for a property. A domain has same characteristics as a set, i.e., the values belonging to a domain are

distinct and their order is immaterial. A domain can contain vectors or matrices. Thus domain D_i is defined as

$$D_i = \{v_i | P_i\}$$

where v_i represents a value, a vector, or a matrix satisfying the predicate P_i.

Attribute. Columns of a two-dimensional table are referred to as attributes. An attribute value can contain relation names and null values.

Example. We consider the sample structural design problem given in section 2.2, to describe these steps.

Step 1. The following entity sets can be identified for the structure

STRUCTURE	(S)
BEAM	(B)
TRUSS	(T)
MEMBRANE-TRI	(TR)
MEMBRANE-QD	(QD)
NODE	(N)
ELEMENT	(E)

Step 2. We can identify the following domains:

STRUCTURE#	Structure identification number (integer)
B#	Beam element identification number (integer)
T#	Truss element identification number (integer)
TR#	Triangular membrane element identification number (integer)
QD#	Quadrilateral membrane element identification number (integer)
NODE#	Node number (integer)
E#	Element number (integer)
EL-TYPE	Element type {BEM2, BEM3, TRS2, TRS3, TRM2, TRM3, QDM2, QDM3}
MATID	Material identification code, e.g., {STEEL•1, STEEL•2, ALUM•5, COMP•1}. It also refers to relation or table of material properties for example, STEEL•1 refers to relation STEEL and material subtype 1
MATPRO	Material property {E,μ,G,...}
CSID	Cross-section type identification code; e.g., {THICK•1, THICK•2, RECT•1, CIRC•5, ISEC•6, LSEC•15}. It also refers to a relation of cross-sectional details. For example, RECT•1, refers to a relation RECT and cross-section subtype 1
CSPRO	Cross-sectional property {H,W,T,R,...}
DOF#	Degrees of freedom numbers
LOAD-TYP	Load type {CONCENTRATED, DISTRIBUTED, TEMPERATURE, ACCELERATION}
X	X coordinate (real)
Y	Y coordinate (real)
Z	Z coordinate (real)
DESCRIPTION	Description (characters)

VEC	Vectors {Integer vector, Real vector, Double precision vectors}
MATX	Matrices {Integer matrices, Real matrices, Double precision matrices}
VECID	Vector identification code ≡ {x•y| x = vector description, y = number}; e.g., FORCE•5, LOAD•10
MAXID	Matrix identification code ≡ {x•y| x = matrix description, y = number}; e.g., EL-STIFF•10, EL-MASS•5

Step 3. The following entity keys are identified

STRUCTURE#	for entity set structure
BEAM#	for entity set beam
TRUSS#	for entity set truss
TR#	for entity set TR
QD#	for entity set QD
E#	for entity set element

Step 4. In the association between entity sets and domain, the entity sets from step 1 are replaced by their primary keys. Attribute names are derived from domain names to provide role identification. The following relations are identified:

For Entity Set STRUCTURE

STRUCTURE (S#, DESCRIPTION, MATXID, MATX)

The structure is identified by a structure number S#. Name of the structure and other details are given in DESCRIPTION. Matrices associated with the structure are identified by MATXID.

For Entity Set BEAM

BEAM (B#, E#, EL-TYP, MATID, E, μ, G, NODE1#, NODE2#, CSID, H, W, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

A beam is identified by a beam number B#. Element number E# uniquely identifies the finite elements of a structure. Attributes NODE1# and NODE2# are derived from domain NODES. Similarly, E, μ, and G are role names for domain MATPRO. CSID identifies the cross-section properties H, W. Vectors and matrices associated with the element are identified through VECID and MAXID, respectively.

Similarly the relations TRUSS, TRM, QDM are as follows:

TRUSS (T#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, CSID, H, W LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

TRM (TR#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, NODE3#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

QDM (QD#, E#, EL-TYP, MATID, E, NODE1#, NODE2#, NODE3#, NODE4#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

NODE (NODE#, X, Y, Z, DOF1#, DOF2#, DOF3#, LOAD-TYP, LOAD#, VICID, VEC)

ELEMENT (E#, NODE#)

From the above example, the following points unique to structural design databases should be noted:

1. The attributes in the example contain relation names (e.g., MATID, CSID). These relations are again association between an entity set; e.g., STEEL and domain of material properties.

2. Null values of domains (which are attributes in the relations) are allowed. For example, in relation TRM, LOAD-TYP and LOAD# may be null if no loads exists.

3. A row and a column intersection may not be a single value; it can be a vector or a matrix.

4.2.3 Reduction to Elementary Relations

In the previous section, we described a method to identify entities, domains and relations to produce a rough conceptual model of the structure. Our idea is to develop a conceptual model which contains all the facts and each fact occurring only once. In order to produce a conceptual data model, we transform the rough model into a better model by using a set of elementary relations which can be defined as:

A relation is irreducible if it cannot be broken down by means of project operations into several relations of smaller degree such that these relations can be joined to reconstitute the original relation. A relation which is not reducible is called an elementary relation.

Elementary relations satisfy the requirement that one fact is recorded in one place. An example of elementary relation is given below:

LOAD (N#, LC#, F_x)
where N# is node number
LC# is load case number
F_x is force in the x-direction

LOAD

N#	LC#	F_x
N1	L1	10.0
N1	L2	15.0
N1	L3	10.0
N2	L2	20.0
N2	L3	20.0
N2	L4	10.0
N3	L3	20.0

Note that this relation describes two facts-load cases, and load values for each node. Now, to see if this is an elementary relation, suppose we split the relation LOAD by means of project operations into following two relations R1 and R2

R1(N#, LC#)

N#	LC#
N1	L1
N1	L2
N1	L3
N2	L2
N2	L3
N2	L4
N3	L3

R2(LC#, F_X)

LC#	F_X
L1	10.0
L2	15.0
L2	20.0
L3	10.0
L3	20.0
L4	20.0

On joining R1 and R2, R1*R2 we get

R3

	N#	LC#	F_X	
	N1	L1	10.0	
	N1	L2	15.0	
→	N1	L2	20.0	← This value comes from <N1,L2> and <L2,20.0>
	N1	L3	10.0	
→	N1	L3	20.0	
→	N2	L2	15.0	
	N2	L3	10.0	
→	N2	L3	20.0	
	N2	L4	20.0	
→	N3	L3	10.0	
	N3	L3	20.0	

The rows marked with → are not in original relation LOAD and hence not correct. Thus, the relation is irreducible, and relation LOAD is an elementary relation. It can be observed in the relation LOAD that the attribute F_X is fully functionally dependent on N# and LC#. N# alone or LC# alone does not determine F_X. Therefore, it is possible to identify such dependencies and establish rules for reducing a relation to an elementary relation. Using the concept of functional dependencies, full functional dependencies, and transitive dependencies, the following steps are identified to form elementary relations.

Step 1. Replace the original relations by other new relations to eliminate any (nonfull) functional dependencies on candidate keys.

Step 2. Replace the relations obtained in Step 1 by other relations to eliminate any transitive dependencies on candidate keys.

Step 3. Go to step 5 if
(a) relation obtained is all key, and
(b) relation contains a single attribute that is fully functionally dependent on a single candidate key.

Step 4. Determine primary key for each relation which may be a single attribute or a composite attribute. Take projections of these relations such that each projection contains one primary key and one non-primary key.

Step 5. Elementary relations obtained.

Example. We can see how these steps are applicable to relations of the example problem in the previous section. Consider the relation TRM.

TRM (TR#, E#, EL-TYPE, MATID, E, NODE1#, NODE2#, NODE3#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

Dependencies	Remarks
TR#	Primary key
TR# → E# also E# → TR#	Secondary key E#
TR# → EL-TYPE	Element type is functionally dependent on TR#
TR# → MATID	Material identification code MATID is functionally dependent on TR#
MATID → E	Material properly E is functionally dependent on MATID
TR# → MATID → E	Material property E is transitively dependent on TR# through MATID
TR# → NODE1#	TR# identifies NODE1#
TR# → NODE2#	
TR# → NODE3#	
TR# → CS-TYP → T	Thickness is transitively dependent on TR# through CS-TYP
TR# → LOAD-TYP	
TR# → LOAD#	
TR# → VECID → VEC	Any vectors, such as load, force, stress vector are transitively dependent on TR# via VECID
TR# → MATXID → MATX	Similarly matrices such as stiffness, mass, are transitively dependent on TR# via MATXID

Reducing relation TRM to elementary relations

Step 1.

ER1 (TR#, E#)
ER2 (TR#, EL-TYP)
ER3 (TR#, NODE1#)
ER4 (TR#, NODE2#)
ER5 (TR#, NODE3#)
ER14 (E#, TR#)

Step 2.

ER6 (TR#, MATID);
ER8 (TR#, CSID);
ER10 (TR#, VECID);
ER12 (TR#, MAXID);
ER7 (MATID, E)
ER9 (CSID, T)
ER11 (VECID, VEC)
ER13 (MATXID, MATX)

Step. 3 The above relations contain single attribute, so go to step 5.

Step. 4. Skip

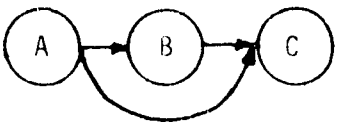
Step 5. ER1 to ER13 are elementary relations.

The steps can be applied to rest of the relations identified earlier to get a set of elementary relations for the sample structural problem.

4.2.4 Determination of Transitive Closure

While deriving a large number of relations for obtaining a conceptual data model, it is possible that some relations might have been missed. In general, it is possible to derive further elementary relations from any incomplete collection of such relations. To explain in a simple way, how such additional relations can be derived, consider two relations ER1(A,B) and ER2(B,C), where

ER1: A → B
ER2: B → C
These implies functional dependencies as



We know that product of functional dependencies leads to transitive dependencies (Vetter and Maddison 1981). Taking product of above functional dependencies, we get

$$A \rightarrow C$$

as indicated above. Therefore, from suitable pairs of elementary relations representing functional dependencies further elementary relations can be derived. Deriving all such relations from initial collection of elementary relations yields a transitively closed collection of elementary relations called transitive closure (Vetter and Maddison 1981). This set of relations includes both derived and original elementary relations. It is complete in the sense that all elementary relations equivalent to transitive dependencies through others are included. There is always a unique transitive closure for a given set of elementary relations. A transitive closure usually contains many redundant relations. We say that an elementary relation is redundant if it is derivable from other relations.

There are problems associated in interpreting relations in transitive closure. For example, consider relations

ER1 (TR#, MATID) where TR# → MATID
ER7 (MATID, E) where MATID → E

Transitive closure for this set yields relation

ER (TR#, E) implies TR# identifies E

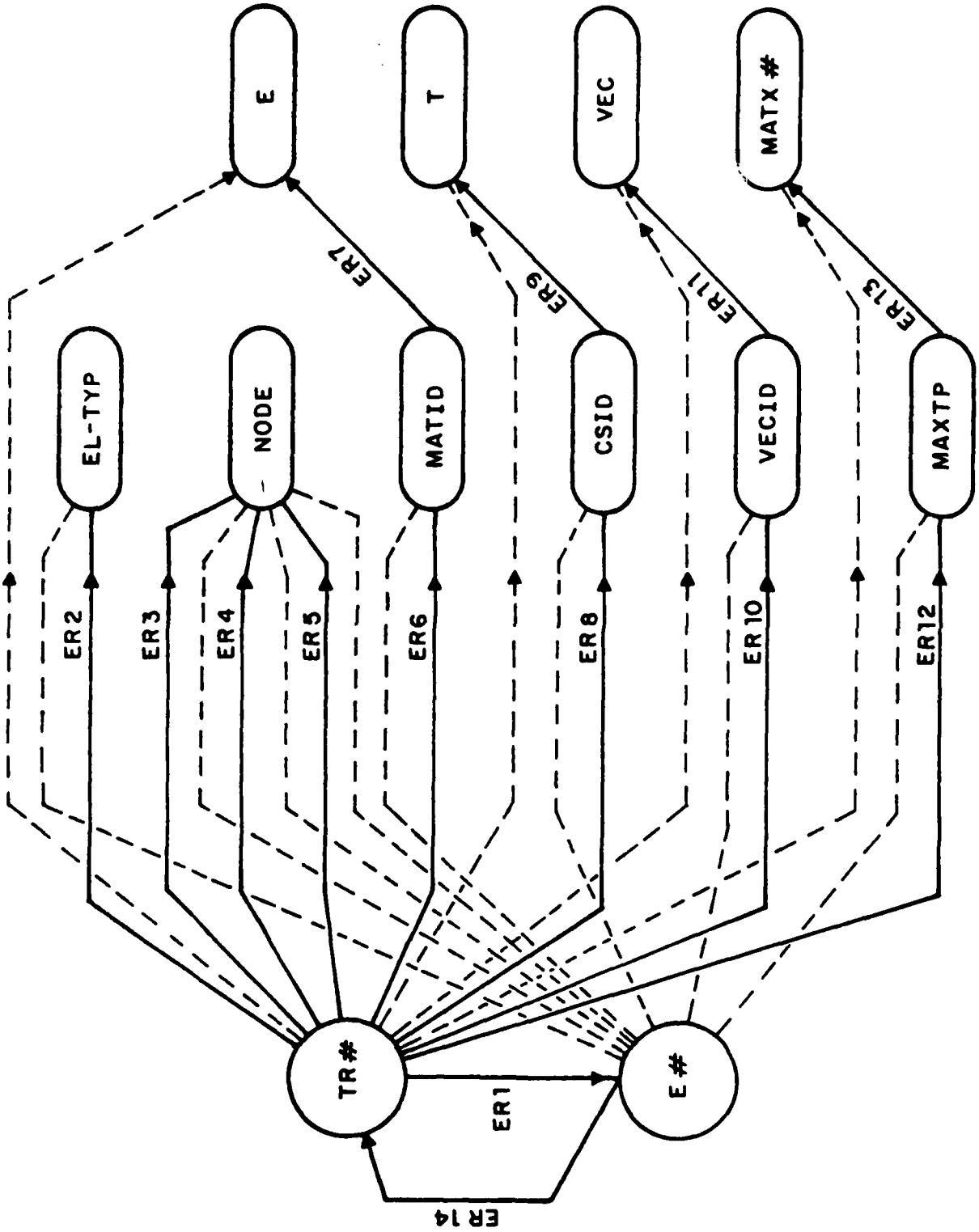
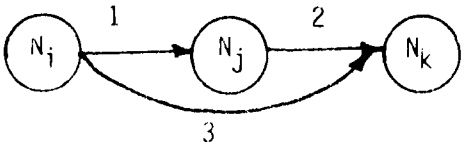


Figure 4.2.1 Digraph Representation of Elementary Relations

However, this relation does not represent true information as material property E is dependent only on material number and not on element number. The relation could be wrongly interpreted. Therefore such semantically meaningless dependencies must be eliminated.

It is possible to determine transitive closure using directed graphs and the connectivity matrix (defined in Section 3.2). The nodes of graph correspond to entity keys and arcs correspond to elementary relations.



Transitive closure is formed by adding arc 3 if arcs 1 and 2 already exist. A diagraph for the elementary relations formed in the previous section is shown in Fig. 4.2.1.

A connectivity matrix for this diagraph is shown in Fig. 4.2.2. In the connectivity matrix, we can see for example, TR# → MATID is indicated by 1 in the corresponding row(3) and column(7) of the matrix. Derived transitive dependence of example TR# → E can be recorded by assigning 1 to C (3,11). Such derived relations are denoted by 1^* in the Fig. 4.2.2 and are denoted by new arcs in dotted lines of Fig. 4.2.1. An algorithm for determining transitive closure is given in Appendix 1.

The transitive closure for the example produces additional dependencies given in Table 4.2.1. We have eliminated meaningless dependencies from the list.

4.2.5 Determination of Minimal Covers

In the previous section, we derived additional elementary relations from a set or original elementary relations of Section 4.2.3. Here, we remove redundant elementary relations to provide a minimal set of elementary relations. Using this process, we obtain one or more minimal covers. A minimal cover is a minimal set of elementary relations from which transitive closure can be derived (Vetter and Maddison 1981).

One may wonder why we first add redundant elementary relations to original list of elementary relation to obtain a transitive closure and in subsequent step we remove redundant elementary relations to obtain a set of minimal cover. Such an approach is justified because (i) minimal cover is not unique, (ii) deriving several alternative minimal covers from a transitive closure guarantees that every possible minimal cover is found, and (iii) we can select a minimal cover that best fits the structural design process needs.

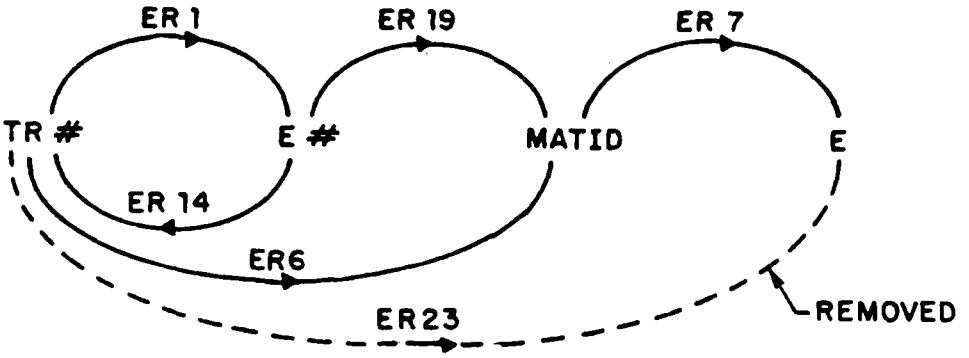
An example for finding a set of minimal cover from the transitive closure derived in previous sections is given in Fig. 4.2.3. A set of minimal cover for this transitive closer is, {ER1, ER14, ER6, ER7}, {ER1, ER14, ER19, ER7}, out of which one set may be choosen to suit our requirement. In the following paragraphs a procedure for determining minimal cover is given (Vetter and Maddison).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	E#	ELTP	TR#	NODE1#	NODE2#	NODE3#	MATID	CSID	VECID	MATXID	E	T	VEC	MATX
1 E#		1*	1	1*	1*	1*	1*	1*	1*	1*				
2 EL-TYP														
3 TR#	1	1	0	1	1	1	1	1	1	1	1*	1*	1*	1*
4 NODE1#														
5 NODE2#														
6 NODE3#														
7 MATID											1			
8 CSID												1		
9 VECID													1	
10 MATXID														1
11 E														
12 T														
13 VEC														
14 MATX														

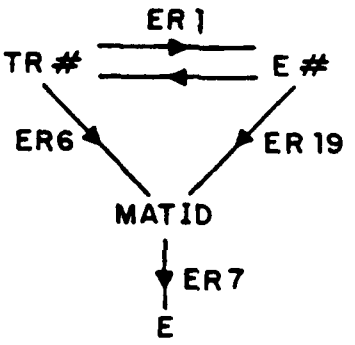
Figure 4.2.2 Connectivity Matrix C for Elementary Data

Table 4.2.1 Transitive Closure for Elementary Relations

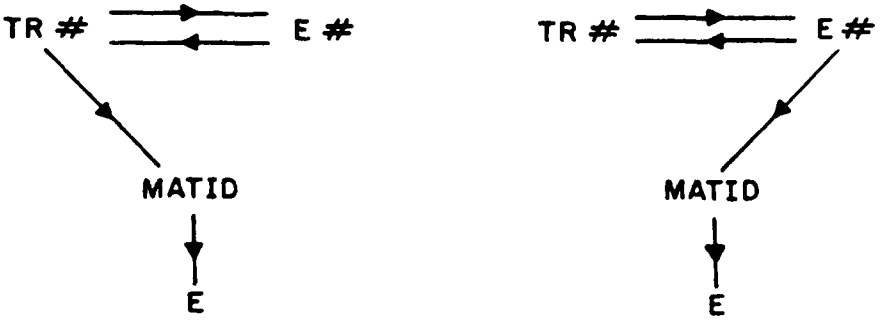
Derived Relations	Dependencies	Composition	Semantically Meaningful
ER15	E# → EL-TYP	E# → TR# → EL-TYP	YES
ER16	E# → NODE1#	E# → TR# → NODE1#	YES
ER17	E# → NODE2#	E# → TR# → NODE2#	YES
ER18	E# → NODE3#	E# → TR# → NODE3#	YES
ER19	E# → MATID	E# → TR# → MATID	YES
ER20	E# → CSID	E# → TR# → CSID	YES
ER21	E# → LOAD-TYP	E# → TR# → LOAD-TYP	YES
ER22	E# → MAXID	E# → TR# → MAXID	YES
ER23	TR# → E	TR# → MAXID → E	NO
ER24	TR# → T	TR# → CS-TYP → T	NO
ER25	TR# → VEC	TR# → VECID → VEC	NO
ER26	TR# → MATX	TR# → MATXID → MATX	NO



(i) Transitive Closure



(ii) Rearrangement



(iii) Minimal Cover

Figure 4.2.3 Digraph Representation of Minimal Cover

Let $ER^z = \{ER_1, ER_2, \ldots, ER_z\}$ represent a list of elementary relation and let $MC\{S_n^m\}$ represent the set of minimal cover (MC). Z and m are number of elements in set ER^z and MC. Let MC be a null set in the beginning. n stands for a running number.

Step 1. Find distance (see Section 3.2 for definition) for each ER in ER^z

Step 2. Remove ER_r from ER^z if it is a longest distance composition of say ER_i and ER_j, i.e., provided that all of

(a) ER_i, ER_j and ER_r belong to ER^z

(b) $ER_r^{\,i} = C^j(Er_i, Er_j)$

(c) ER_r possesses maximum distance

Call the remaining set ER_n^{z-1}. If no element is removed from ER^z, place ER^z in MC and terminate. In general one can find

$$ER_1^{z-1}, ER_2^{z-1}, \ldots, ER_n^{z-1}$$

Step 3. Repeat Step 2 for each ER_n^{z-1} and obtain ER^{z-2}, then add new element to MC

Step 4. Repeat Step 3 until no element can be removed and then terminate; get $MC\{S_n^m\}$

We demonstrate the above procedure for the example considered earlier. We have $ER^5 = \{ER_1, ER_{14}, ER_6, ER_{19}, ER_7\}$ and these are renumbered as $\{ER_1, ER_2, ER_3, ER_4, ER_5\}$. The distances are

1 ER_1: distance $d_1 = 1$

2 ER_{14}: $d_{14} = 1$

3 ER_6: $C(ER_1, ER_{14})$ $d_6 = 2$

4 ER_{19}: $C(ER_{14}, ER_6)$ $d_{19} = 2$

5 ER_7: -- $d = 3$ (E# → ER# → MATTYP → MAT#)

They are renumbered as $\{d_1, d_2, d_3, d_4, d_5\}$. Derivation of minimal cover is given in Table 4.2.2. After 1st iteration MC (i.e., the set of minimal covers) consist of

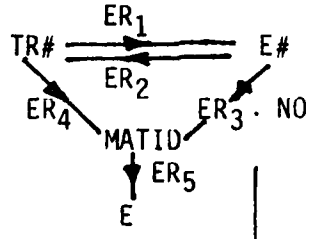
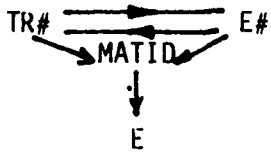
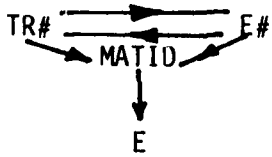
$$MC\{ER_1^4, ER_2^4\} \text{ with}$$

$$ER_1^4 = \{ER_1, ER_2, ER_4, ER_5\} = \{ER_1, ER_{14}, ER_6, ER_7\}$$

$$ER_2^4 = \{ER_1, ER_2, ER_3, ER_5\} = \{ER_1, ER_{14}, ER_{19}, ER_7\}$$

These are two alternate minimal covers.

Table 4.2.2 Deriving Minimal Cover

Iteration	ER^z	ER	$C(ER_i, ER_j)$	d_r	E_r Removed	ER_n^{z-1}	Digraph (Before Removed)	MC
0	ER^5	ER_1	--	1				NO
		ER_2	--	1				
		ER_3	$C(ER_1, ER_2)$	2	+	ER_1^4		
		ER_4	$C(ER_2, ER_4)$	2	+	ER_2^4		
		ER_5	--	3	is not composed of other relations			
1	ER_1^4	ER_1	--	1				YES
		ER_2	--	1				
		ER_4	--	2				
		ER_5	--	3				
1	ER_2^4	ER_1	--	1				YES
		ER_2	--	1				
		ER_3	--	2				
		ER_5	--	2				

The above procedure can be applied to other transitive closures derived in the previous section. Thus, we can get further sets of minimal covers. Each minimal cover is a non-redundant list of elementary relations and is an appropriate conceptual model of the structural design data.

4.3 Internal Model

Internal model deals with the logical organization of data to be stored on physical storage devices. The internal model specifies which attributes have to be combined together as a unit and how the relationship between these units are represented. Here, we are concerned about finding an internal model which supports the conceptual model in an efficient way. Also, the internal model must be consistent with the conceptual model.

One can build an internal model by storing all the elementary relations that represent a conceptual model in a database. However, such an approach would require a large number of accesses to the database to get data about an entity and would be highly inefficient. Another approach would be to build an internal model by means by a wider n-ary relations. These n-ary relations have to be consistent with the conceptual model and they have to follow certain rules. Any storage and update operations (insert, modify, delete) must not lead to inconsistency in data. To avoid anomolies in storage and update operations normalization procedure are adopted. In the following paragraphs, we describe a methodology for building an internal model based on normalization procedures. Examples from structural design are used to describe the procedures.

Design of an internal model to support element stiffness matrix generation is described here. Methodology for designing internal models for other structural analysis and design process would follow similar steps. We assume that a conceptual model for element stiffness generation is already available. Our aim is to produce an internal model that is consistent with the conceptual model given by the following elementary relations:

ER1 (TR#, EL-TYP)
ER2 (TR#, NODE1#)
ER3 (TR#, NODE2#)
ER4 (TR#, NODE3#)
ER5 (TR#, MATID)
ER6 (MATID, E)
ER7 (TR#, CSID)
ER8 (CSID, T)
ER9 (NODE#, X)
ER10 (NODE#, Y)
ER11 (NODE#, Z)
ER12 (TR#, MATXID)
ER13 (MATXID, MATX)
ER14 (E#, NODE#)
ER15 (TR#, E#)

Data needed for generation of element stiffness matrix are derived from various domains and represented in a single relation TRM-D as shown in Fig. 4.3.1. Our main intention is to get all the data required for generation of stiffness matrix for a triangular membrane element in one access or minimum number of accesses.

It can be observed from the relation of Fig. 4.3.1 that it is not in first normal form. Therefore, this unnormalized relation should be replaced by a semantically equivalent relation in 1NF as shown in Fig. 4.3.2.

The following points have to be noted for normalization procedure in structural design context:

1. Row and column intersection of a relation in 1NF is atomic (i.e., consists of single values). Exception to these rules are vectors and matrices and they are considered to be atomic. In Figure 4.3.2, a set of values of MATX is identified as a unit.

2. Values for attributes are also derived from a non-simple domain. A non-simple domain is one which contains elements that themselves are relations. In Figure 4.3.2, STEEL refers to another relation which contain material properties.

The advantage of 1NF over the unnormalized relation are that operations required for application programs are less complicated and easy to understand.

The elementary relations identified earlier contain all the information required for generation of element stiffness matrices. This information should be reflected in some way in the internal model that is to be developed. There by we can ensure that internal model is consistent with the conceptual model. Internal model of Fig. 4.3.2 has all the attributes providing information for generation of element stiffness matrix. To check the consistency of this model, first we identify the key attributes. Candidate keys are compound, consisting of (E#, NODE#) and (TR#, NODE#). Primary key is selected as (TR#, NODE#). Secondary keys are TR#, E#, MATID, CSID, NODE#, and MATXID. These key attributes of the relation are consistent with those in the elementary relation. Secondly, we need to identify whether all the attributes in internal model and dependencies between them are consistent with the conceptual model. It can be observed that attributes NODE1#, NODE2# and NODE3# do not appear in the relation. Therefore, these three attributes should be included in the relation. Relation TRM-D is now written as

TRM-D (TR#, E#, EL-TYP, E, CSID, T, NODES#, NODE1#, NODE2#,
NODE3#, X, Y, X, MATXID, MATX)

The functional dependencies reflected by elementary relations ER1 to ER15 are satisfied in the internal model with the values shown in the Fig. 4.3.2. Therefore, at this instant the internal model is consistent with the conceptual model. However, it would be no longer consistent, if arbitrary changes in the values of the table are made. Also, note that many values in the relation TRM-D are redundant. These inconsistencies and redundancies occur because of the following anomalies in the 1NF:

Domain TR# | Domain MATID | Domain CSID | Domain NODES | Domain X,Y,Z | Domain MATXID

Domain EL-TYP | Domain Material Property | Domain Cross-sectional Property | Domain E# | Domain MATX

TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL•5	10^8	THICK•6	0.1	5 7 8	1. 3. 2.	5. 4. 6.	7. 8. 3.	STF•1	[:::]
2	16	TRM3	ALUM•4	$0.9x10^7$	THICK•4	0.2	12 14 18	5. 7. 3.	9. 4. 5.	8. 1. 8.	STF•2	[:::]

Figure 4.3.1 A Tentative Internal Model

TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL•5	10^8	THICK•8	0.1	5	1.	5.	7.	STF•1	[:::]
1	15	TRM3	STEEL•5	10^8	THICK•8	0.1	7	3.	4.	8.	STF•1	[:::]
1	15	TRM3	STEEL•5	10^8	THICK•8	0.1	8	2.	6.	3.	STF•1	[:::]
2	16	TRM3	ALUM•4	$0.9x10^7$	THICK•4	0.2	12	5.	9.	8.	STF•2	[:::]
2	16	TRM3	ALUM•4	$0.9x10^7$	THICK•4	0.2	14	7.	4.	1.	STF•2	[:::]
2	16	TRM3	ALUM•4	$0.9x10^7$	THICK•4	0.2	18	3.	5.	8.	STF•2	[:::]

Figure 4.3.2 Relation TRM-D in 1NF

1. INSERT operations: User cannot store details concerning a finite element without knowing at least one node of the element. The reason is that one part of the primary key (E#,NODE#), i.e, NODE# is not known.

2. DELETE operations: If user deletes a particular material type - MATID, the database automatically looses all the data about those finite elements using the material. Similarly, if user deletes a particular finite element, then database looses data about a particular material.

3. MODIFY operations: To change information about a particular element number, all rows containing that element number have to be modified. Otherwise functional dependency MATID → E will not be valid any more.

Therefore it is not desirable to use the relation in Fig. 4.3.2 to represent the internal model. Modification of this relation to 2NF is necessary to avoid these anomalies in the storage operations. For this purpose, first we need to identify non-key attributes of TRM-D and check their dependence on the candidate keys. The non-key attributes are EL-TYP, E, T, X, Y, Z, MATX (refer to definition of full functional dependency). For example, consider the non-key attribute EL-TYP:

$$E\#,\ NODE\# \rightarrow EL\text{-}TYP$$
$$E\# \rightarrow EL\text{-}TYP$$
$$NODE\# \not\rightarrow EL\text{-}TYP$$

Therefore, $E\#,\ NODE\# \not\Rightarrow EL\text{-}TYP$

Thus, EL-TYP is not fully functionally dependent on (E#,NODE#). Note that E is transitively dependent on E# through MATID. Similar argument leads to

$E\#,\ NODE\# \not\Rightarrow E,\ T,\ X,\ Y,\ Z$ or MATX.

Therefore, relation TRM-D should be converted into a set of semantically equivalent relations as follows:

TRM-D1 (TR#, E#, EL-TYPE, NODE1#, NODE2#, NODE3#, MATID, E, CSID, T, MATXID, MATX)
TRM-D2 (NODE#, X, Y, Z)
TRM-D3 (E#, NODE#)

The above three relations TRM-D1, TRM-D2 and TRM-D3 are all in 2NF because first two relations do not possess any compound candidate key and third relation has all keys. Note that by splitting the relation, TRM-D no information is lost and they still are consistent with the conceptual model. However TRM-D1 relation is not still satisfactory. It can lead to anomolies in storage operations as follows:

1. INSERT operation: It is not possible to store the fact that a particular material - MATID has a property - E without knowing at least one finite element using the material.

2. DELETE operation: If only one element is using a particular material and if that element is deleted, we loose all information about the material.

3. MODIFY operation: If several finite element use a particular material and if the property of the material is changed, then modification must be done to all the rows of the material used by those elements.

Therefore it is not feasible to use TRM-D1 relation in the internal model. Modification of this relation is necessary to 3NF to avoid anomalies in storage operation. Non-key attributes must be non-transitively dependent on candidate keys to avoid these anomolies. It can be observed from the relation TRM-D1 of Fig. 4.3.3 that attributes E, T, and MATX are transitively dependent on TR# through MATID, CSID, and MATXID, respectively. Removing these transitive dependencies, we get the following relation:

TRM-D4 (TR#, E#, EL-TYP, NODE1#, NODE2#, NODE3#, MATID, CSID, MATXID)
TRM-D5 (MATID, E)
TRM-D6 (CSID, T)
TRM-D7 (MATXID, MATX)

The above three relations together with TRM-D2 and TRM-D3 constitute the internal model for element stiffness matrix generation purpose. This internal model is consistent with the conceptual model identified earlier. Also, note that number of relations in the internal model is only 6 as compared to 15 elementary relations in the conceptual model.

In summary, the following steps are necessary to derive an internal model that is consistent with the conceptual model. Normalization procedures have to be adopted at each step to reduce redundancy, to eliminate undesired anomolies in storage operation and to ensure integrity of the stored values in the database. At each step unsatisfactory relations are replaced by others.

Step 1. Form relations with attributes derived from a set of domains.

Step 2. Eliminate multiple values at row-column intersection of relation table. Vectors and matrices are considered to be single data items for this step.

Step 3. Result of Step 2 is relations in 1NF. Take projections of 1NF relations to eliminate any nonfull functional dependencies and get a relation in 2NF.

Step 4. Take projection of relations obtained in Step 3 to eliminate transitive dependencies to form relations in 3NF. Thus a set of relations in 3NF is the internal model.

Implementation of these relations in actual physical storage is dependent on the database management system. Rows of a relation correspond to stored record occurrences in a physical storage (disk) device. Records in the disk may be stored successively or may be connected by a links or index to records may be provided. Details of physical storage are discussed in the next two chapters.

TRM-D1

TR#	E#	EL-TYP	NODE1#	NODE2#	NODE3#	MATID	E	CSID	T	MATXID	MATX
1	15	TRM3	5	7	8	STEEL•5	10^8	THICK•8	0.1	STF•1	[::::]
2	16	TRM3	12	14	18	ALUM•4	$0.9x10^7$	THICK•4	0.2	STF•2	[::::]

TRM-D2

NODE#	X	Y	Z
5	1.	5.	7.
7	3.	4.	8.
8	2.	6.	3.
12	5.	9.	8.
14	7.	4.	1.
18	3.	5.	8.

TRM-D3

E#	NODE#
15	5
15	7
15	8
16	12
16	14
16	18

Figure 4.3.3 Relations in 2NF

4.4 External Model

One of the important requirements of a database is to provide facility for data retrieval by different application programs depending on their needs. Different application programmers can have different views of a database. Data structure as seen by an application program or interactive user is called an external data model. Generally, users perspective of a database is only a subset of the actual contents of the database. Data retrieved from actual physical storage in the database undergoes transformation till it reaches the user. Transformation involves rearrangement of data from internal level to external level into a form acceptable to the application program. In the following paragraphs, considerations required while designing an external model that is suitable to structural design applications are given.

We have discussed the various data models - hierarchical, network and relational in Chapter 3. Some of these models could be choosen for use as external models. Choice of the external model depends on consideration of simplicity of use for applications and database management system's capability. Relational model offers excellent facility for providing simple and easy view of data for application programmers and interactive users. Tabular form of data offered by the relational model is convenient to represent most of the structural design data. Hierarchical model is suitable to organize data of a hypermatrix. A hypermatrix consists of a number of submatrices organized at various levels. Network model is more general than hierarchical model, but the complicated structure of the model makes it difficult to use. Therefore, relational and hierarchical models can be choosen to organize structural design data. As stated earlier, the choice for the external data model also depends on the capability of the database management system (defined in later chapters) available to support users view of data. It is assumed that a database management system is available to support relational and hierarchical models for purpose of our discussion.

Some constraints have to be observed while designing an external model. Constraints arise while rearranging data from internal data structure to an external data structure. An important constraint is that internal data structure must be consistent with the conceptual data structure. Any retrieval and storage operations specified on external model must be correctly transformed into corresponding operations on the internal model and at the same time data must be consistent with the conceptual data model. Also design of the external model must fit the database management system capability. An example of how an external model is derived from an internal model is given below.

Suppose a particular user would like to know the coordinates of nodes of each triangular finite element for generation of element stiffness matrices. This means that the external model:

EL-CORD (TR#, E#, EL-TYPE, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)

has to be provided for that particular user. Note that the external view EL-CORD contains data items from two different relations -- TRM-D4, TRM-D2 (refer Section 4.3). Therefore, a procedure is required to transform internal data model (relations TRM-D4, TRM-D2) to the external data model (relation EL-

CORD). Data from TRM-D4 and TRM-D2 have to be rearranged to obtain relation EL-CORD. Procedure for rearrangement can be formulated by using JOIN and PROJECT operations as follows:

```
TRM-A (TR#, E#, EL-TYP, NODE1#) ← TRM-D4
TRM-B (TR#, E#, EL-TYP, NODE2#) ← TRM-D4
TRM-C (TR#, E#, EL-TYP, NODE3#) ← TRM-D4
TRM-D (TR#, E#, EL-TYP, X1, Y1, Z1) = TRM-A*TRM-D2
TRM-E (TR#, E#, EL-TYP, X2, Y2, Z2) = TRM-B*TRM-D2
TRM-F (TR#, E#, EL-TYP, X3, Y3, Z3) = TRM-C*TRM-D2
ELCORD (TR#, E#, EL-TYP, X1, Y1, Z1, X2, Y2, Z2,
       X3, Y3, Z3) = TRM-D*TRM-E*TRM-F
NOTE:  ← indicates PROJECT; * indicates JOIN
```

The above procedure (algorithm) yields EL-CORD relation. It can be seen from the algorithm that we did not modify the original relations TRM-D4 and TRM-D2 to retrieve the data required for a particular inquiry. The relations TRM-D4 and TRM-D2 are still consistent with the conceptual model. Therefore, pure retrieval operations for rearrangement of data does not cause any inconsistency in data values.

Now, consider the reverse process of transforming external data structure to internal data structure. Suppose, a particular user wants to insert the nodal coordinates of a finite element using the external view EL-CORD. Here, relation EL-CORD has the only key TR# and has no reference to the node number to which the element is connected. Insertion is not consistent with the conceptual model which requires that coordinate of nodes which are dependent on keys NODE#. This restriction is also reflected in the internal model -- TRM-D2 which requires NODE# as key values for insertion. Therefore, the transformation of relation EL-CORD into internal model is not possible. From this example, it follows that there are restrictions for rearranging data from external model to internal model.

4.5 Numerical Model

In finite element analysis and structural design optimization, we encounter problem of storage of large order matrices. These matrices are generally banded and sparse, and require careful consideration in organizing them in databases. This special nature of matrix data is unique to design database and therefore no attempts have been made to study this aspect in business database management area. However, a few matrix schemes have been implemented on disk files, but they are highly tailored to meet only specific application program needs and not suitable for general use. Consequently, there is a need for the development of a generalized and a new user-friendly technique to deal with large order matrices. In this section, we discuss various types of large order matrices and develop a suitable methodology for organizing them in a database.

4.5.1 Identification of Matrices

Various types of matrices are identified and defined below. Note that matrices considered for our purpose are of large order which implies a matrix

A(m,n) where m and n about 1000 or more. For the purpose of our discussion, matrices are grouped into five types and are referred by the type numbers.

(i) Square Matrix A

$A \equiv a(i,j) \quad i = 1, 2, \ldots n, \quad j = 1, \ldots m$

A square matrix A is symmetric if $A = A^T$

A square matrix A is diagonal if

$a(i,j) \neq 0 \quad$ for $i = j$

$a(i,j) = 0 \quad$ for $i \neq j$

A square matrix A is upper triangular if

$a(i,j) \neq 0 \quad$ for $i \leq j$

$a(i,j) = 0 \quad$ for $i > j$

A square matrix A is lower triangular if

$a(i,j) \neq 0 \quad$ for $i \geq j$

$a(i,j) = 0 \quad$ for $i < j$

(ii) Banded Matrix A

$a(i,j) = 0 \quad$ for $|i-j| > m$

where $m << n$; n = matrix size

$a(i,j) \neq 0 \quad$ for $|i-j| \leq m$

Consider $b_j = 1 + (j-1)$ for all i, where j is the column number for last nonzero entry in row i, then semi-band width $B = \max b_i$ (refer to Fig. 4.5.1).

(iii) Hypermatrix H

$H \equiv h(k,\ell) \quad k = 1, \ldots p, \quad \ell = 1, \ldots p$

where h(k,L) $\equiv$ square nonnull matrix A for some, k,ℓ

$\equiv$ square null matrix A for some k,ℓ

and $A \equiv a(i,j) \quad i = 1, \ldots m, \quad j = 1, \ldots m$

A is known as submatrix of H. k and ℓ are known as hyper-rows and hyper-columns (refer to Fig. 4.5.2).

H is upper triangular if

$h(k,\ell) \equiv A$ for $k \leq \ell$

$\equiv 0$ for $k > \ell$

H is lower triangular if

$h(k,\ell) \equiv A$ for $k \geq \ell$

$\equiv 0$ for $k < \ell$

(iv) Sky-line Matrix S

For symmetric upper triangular matrix

$S \equiv a(i,k) \quad i = 1, \ldots n, \; k = 1, \ldots n$

m_j = row number of first nonzero element in column j; m_j, $j = 1, \ldots m$ define the skyline.

$(j-m_j)$ = column height

$a(i,k) = 0 \quad$ for $k > (j-m_j)$; (refer to Fig. 4.5.3).

(v) Sparse Matrix P

$P \equiv a(i,j) \quad i = 1, \ldots m, \; j = 1, \ldots n$

P is sparse if

$a(i,j) = 0$ for most values of i and j. As a rule of thumb, only about 5 to 20% of the matrix contains nonzero values at scattered locations in the sparse matrix.

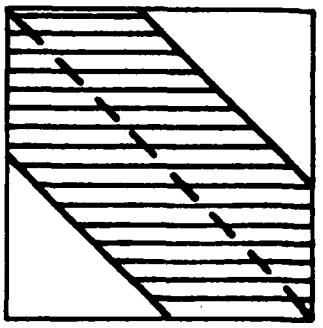


Figure 4.5.1 Banded Matrix

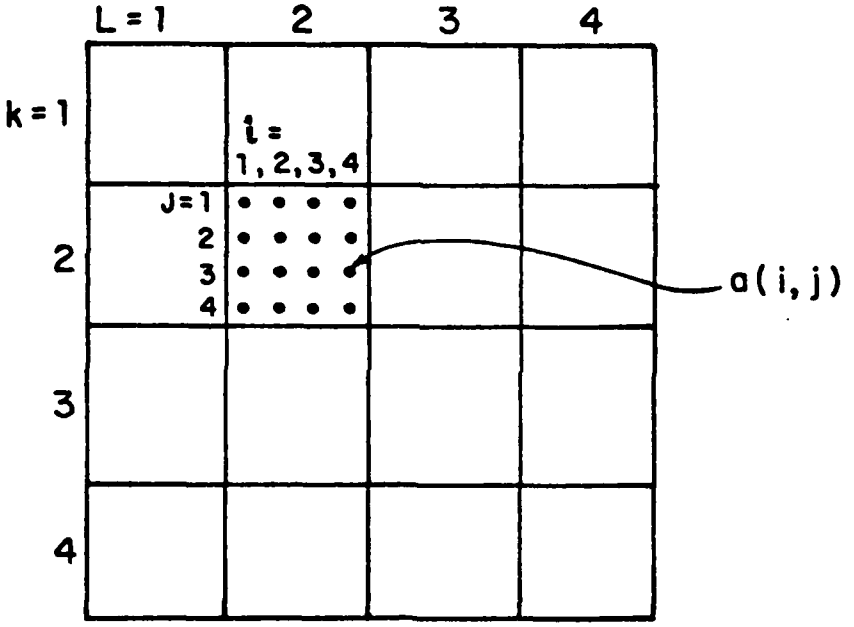


Figure 4.5.2 Hyper Matrix

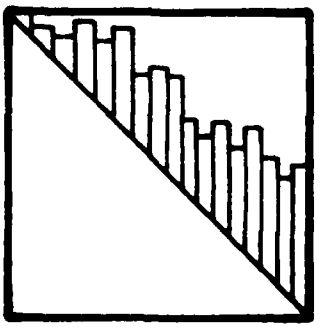


Figure 4.5.3 Skyline Matrix

4.5.2 Methodology for Design of a Numerical Model

We identified various types of matrices commonly encountered in finite element analysis and design optimization procedures. It is necessary to establish a methodology for organizing these matrices in a database. A numerical model is proposed in this section which consists of conceptual, internal and external views of large matrices. Recall that a conceptual view represent inherent nature of data independent of any computer constraints. Therefore it is necessary to first study true nature of a large order matrix. Later, the internal representation of a matrix can be considered to deal with storage efficiency, processing sequence, matrix operations, and flexibility of data modification. Since different applications and users view the same matrix in different form, suitable external views have to be provided.

Conceptually, a matrix is a two-dimensional array of numbers. These numbers appear in a certain pattern; e.g., square, sparse, symmetric diagonal, banded, lower triangular form, upper triangular form, unitary form, tridiagonal form, hyper matrix form and skyline form. A matrix is uniquely identified by a name. Rows and columns of the two-dimensional array are used for identification of data elements in the matrix. A conceptual view of a matrix can be represented by the following elementary relations:

ER1 (NAME, MATRIX TYPE)
ER2 (NAME, NUM-OF-ROWS)
ER3 (NAME, NUM-OF-COLUMNS)
ER4 (NAME, ROW, COLUMN, DATA-ELEMENT-VALUE)
ER5 (NAME, NUM-OF-HYPER ROWS)
ER6 (NAME, NUM-OF-HYPER COLUMNS)
ER7 (NAME, HYP-ROW, HYP-COLUMN, ROW, COLUMN, DATA-ELEM-VALUE)
ER8 (NAME, BAND-WIDTH)
ER10 (NAME, SUB-MAT-ROW-SIZE)
ER11 (NAME, SUB-MAT-COLUMN-SIZE)
ER12 (NAME, VECTOR OF SKYLINE-HEIGHT)
ER13 (NAME, HYP-ROW, HYP-COLUMN, NULL-OR-NOT)

The attributes of these elementary relations are self descriptive. These elementary relations completely define a matrix and provide the conceptual structure of the matrix. Note that the data values assigned to a matrix do not depend on whether a matrix is in banded, skyline or hyper matrix form. But they are located using on the row and column number of a matrix. Therefore, additional information such as banded, skyline, hyper matrix, bandwidth, and submatrix size are useful mainly to take advantage of the special nature of a matrix for storage and computational purpose.

Internal (storage) structure for large order matrices have to be developed which is consistent with the conceptual structure. The elementary relations defined above could be stored in a database, but it would require an awful number of accesses to get the required matrix data. Therefore, storage schemes have to be developed based on efficiency considerations. Also, storage space consideration is important to save disk space. Special nature of matrix, i.e., is sparse, dense, symmetric, should be used to provide storage efficiency. We can classify various matrix types considered in the previous section into two basic types - sparse and dense. Note that banded or

diagonal matrices are not to be mistaken as sparse. Many possible storage schemes are available to store dense and sparse matrices. First, we consider storage of large order dense matrices.

Conventional storage schemes -- row-wise, column-wise, submatrix-wise are useful for storing dense matrices. Row and column storage are considered to be similar for purpose of our discussion. Thus, out of these storage schemes, only two schemes -- row-wise and submatrix wise are considered for evaluation. Figure 4.5.4 shows the row and submatrix storage schemes. Again, it is stressed here that we are considering the internal storage schemes and not the users view of a matrix - such as row-wise, column-wise, submatrix wise, skyline wise, or upper-triangular. Choice between these two storage schemes should be based on consideration of several aspects - storage space, processing sequence, matrix operation, page size, flexibility for data modification, ease of transformation to other storage schemes or user's views, number of addresses required to locate rows or submatrices, and availability of database management system support. These aspects are considered in detail below.

Storage Space Row storage scheme can be used for square, banded and skyline matrix types. However, this scheme is not appropriate for hypermatrix. Symmetric, triangular, and diagonal properties of square matrix can be used in saving storage space if variable length of rows is used. Similar schemes can be used for banded and skyline matrices to store data elements that appear in a band or skyline column. Submatrix storage can be used for all matrix types. Submatrix storage is most appropriate for hypermatrix data. Both schemes have disadvantages when zero elements within a row or submatrix have to be stored.

Processing Sequence Row storage requires assembly of matrices, storage and retrieval be made only row-wise. This becomes inefficient if row-wise processing cannot be made. Submatrix approach is suitable for all types of processing sequence -- row-wise, column-wise, or in any arbitrary order.

Matrix Operations Operations such as transpose, addition, multiplications and solutions of simultaneous equations are frequently carried out at various stages of structural design. Row storage scheme is highly inefficient for matrix transpose when column-wise storage is required. During multiplication of two matrices A and B, a column of B can only be obtained only by retrieving all of rows of B. Therefore, row storage scheme become inappropriate for such operation. However, submatrix storage scheme does not impose any such constraints in matrix operation, thus provides a suitable internal storage scheme.

Page Size A page is a unit or block of data stored or retrieved from memory to disk. A more detailed definition will be given in the next chapter. For a fixed page size, only a number of full rows or a number of full submatrices together with fractional parts of them can be stored or retrieved at a time. It is clear that fragmentation of rows or submatrices takes place depending upon the size of rows or submatrices. Large row size will overlap more than one page in memory and cause wastage of space. Submatrix scheme has the advantage of providing flexibility in choosing submatrix size to minimize fragmentation of pages.

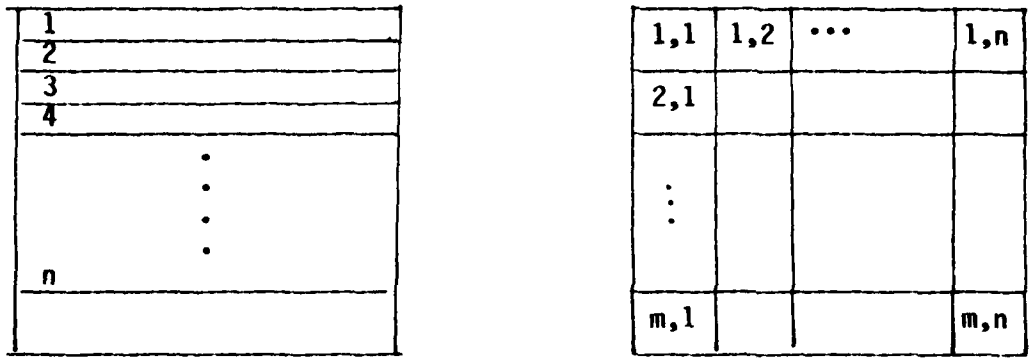


Figure 4.5.4 Row and Submatrix Storage Schemes

Flexibility for Data Modification For modifications of rows of a matrix both row and submatrix storage schemes are suitable. But row scheme would be more efficient than submatrix storage scheme. For modifications of a few columns of a matrix, row storage scheme requires a large number of I/O.

Transformation to Other Schemes Submatrix storage scheme requires minimum number of data access to transform to column-wise storage scheme.

Address Required Submatrix storage requires less number of addresses to locate data than row storage scheme provided submatrices are reasonably large.

Thus, from various aspects considered for choice of storage scheme, submatrix storage scheme has clear advantage over the row storage scheme. Hence submatrix storage scheme can be used for internal storage of large order matrices in a database.

In order that internal storage scheme be consistent with the conceptual model, we need to store additional information about the properties of the matrix. Those additional informations are given by the elementary relations ER1, ER2, ER3, ER5, ER6, ER9 to ER13. They can be combined together and stored in a relation with key attribute NAME. Relations required for internal storage are indicated in Fig. 4.5.5.

So far we considered schemes for internal organization of large matrices. Since different users view the same matrix in different forms - banded, skyline, hypermatrix, triangular, diagonal, it is necessary to provide external views to suit individual needs. Unit of transactions on various views of a matrix may be row-wise, column-wise, submatrix-wise or data element wise. Internal scheme is submatrix-wise, where as external view need not be submatrix wise. Therefore, transformation procedures are necessary to convert the internal matrix data into the form required for a particular user. Such a transformation is schematically indicated in the Fig. 4.5.6.

Next, we consider sparse matrix storage scheme. Several storage schemes have been suggested by Pooch (1973) and Daini (1982). They are bit-map scheme, address map scheme, row-column scheme, and threaded list scheme. Out of these row-column scheme is simple and easy to use. Also, row-column scheme can be easily incorporated into relational model. Therefore, this scheme can be considered for storing sparse matrices encountered in design sensitivity analysis.

Row-column storage scheme consists of identification of row and column numbers of nonzero elements of a sparse matrix and storing them in a table. This scheme provides flexibility in modification of data. Any nonzero value generated during a course of matrix operation can be stored or deleted by simply adding or deleting a row in the stored table. The row-column scheme is schematically shown in Fig. 4.5.7.

External view of row-column storage scheme can be provided through suitable transformation procedures. An external view of this scheme is shown in Fig. 4.5.8.

NAME	MATRIX TYPE	NO. OF ROWS	NO. OF COLUMN	SUBMATRIX ROW SIZE	SUBMATRIX COLUMN SIZE	BANDWIDTH	VECTOR OF SKYLINE HEIGHT

NAME	HYP-ROW NO.	HYP-COLUMN NO.	NULL or NOT	SUBMATRIX
A	1	1		[⋮⋮⋮]
A	1	2		[⋮⋮⋮]
...		...	...	...

Figure 4.5.5 Relations for Matrix Storage

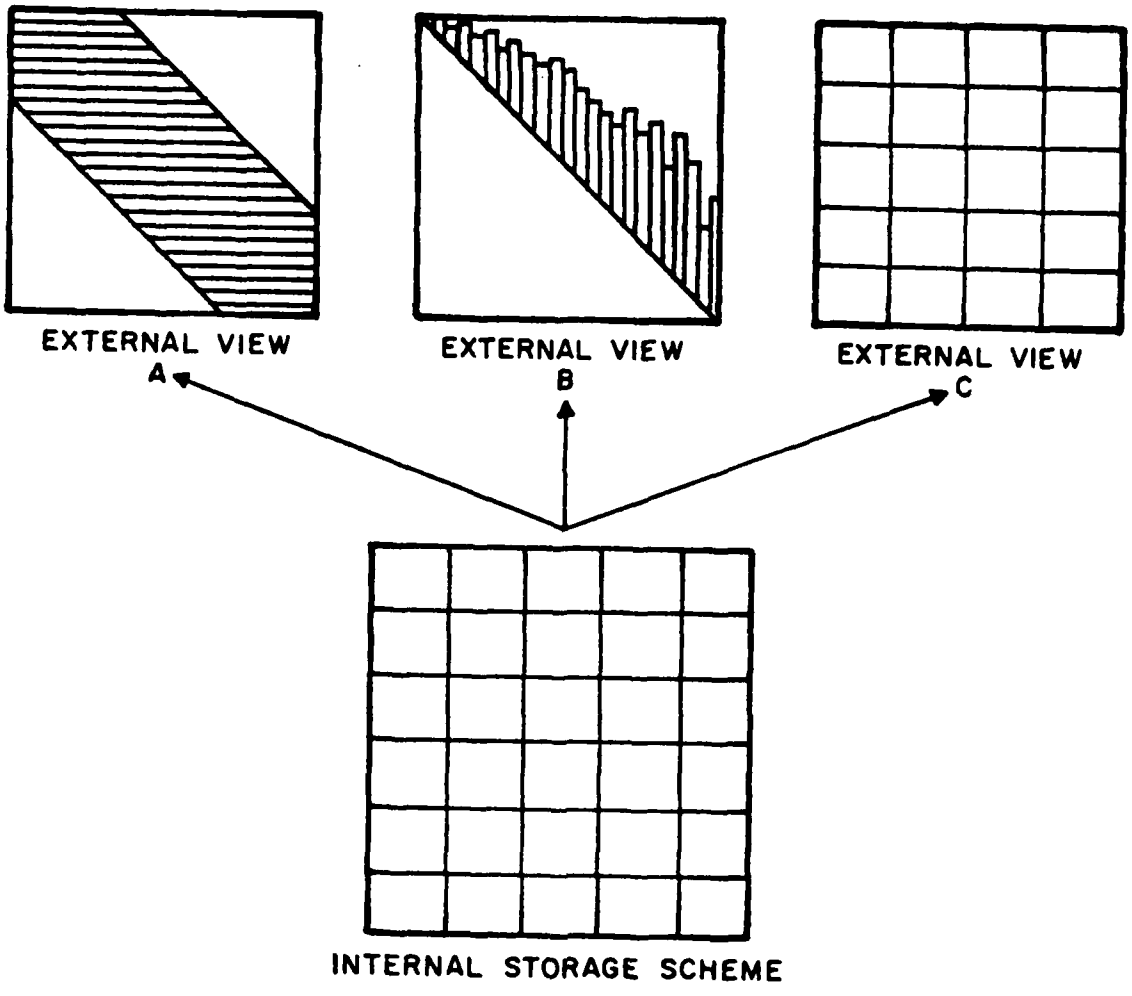


Figure 4.5.6 Transforming Internal Storage to External Views

ROW NO	COLUMN NO	VALUE

Figure 4.5.7 Row-Column Storage Scheme (Internal)

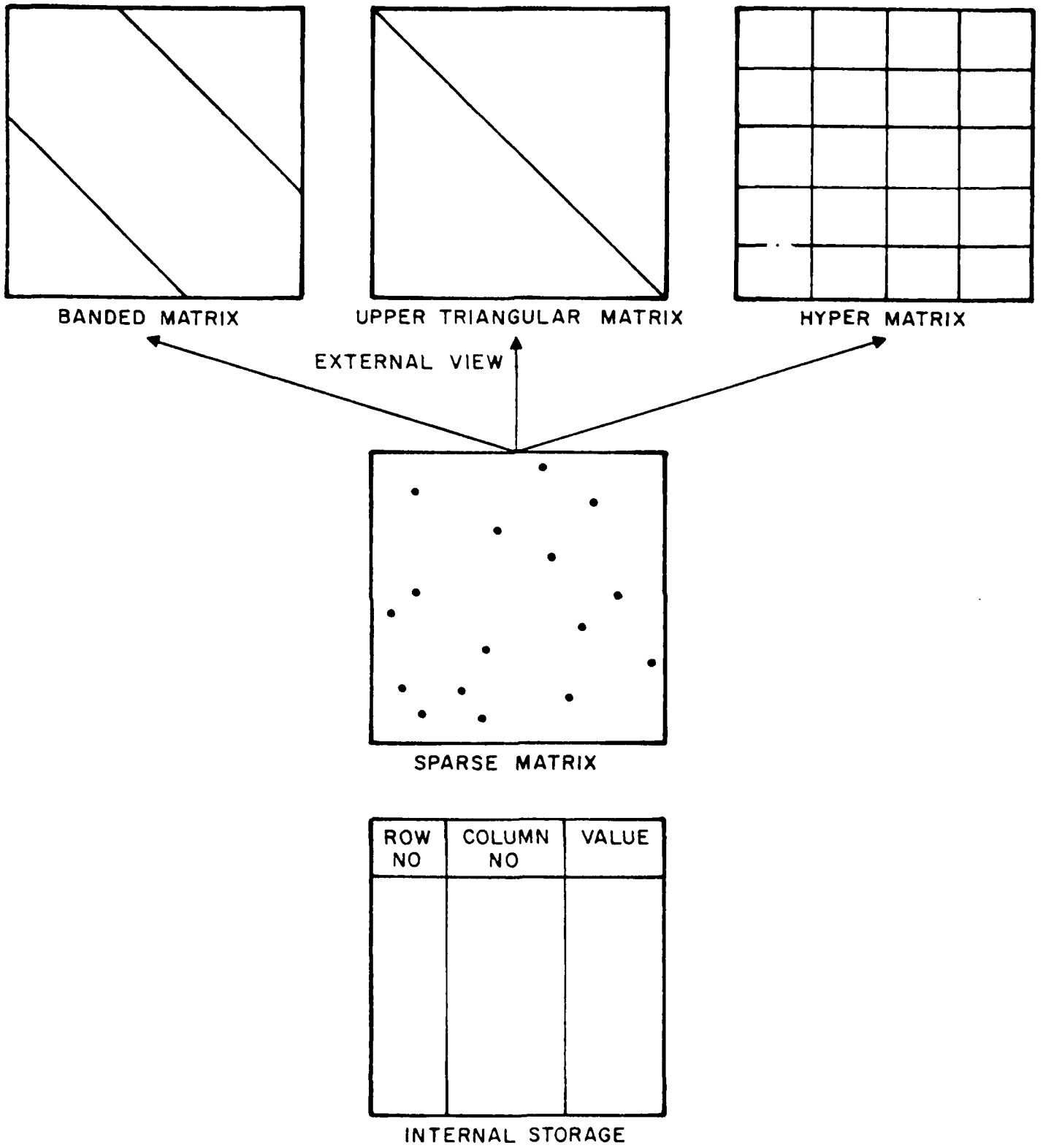


Figure 4.5.8 External View of Sparse Matrix

4.6 Algorithmic Model

So far we considered various methods and procedures for data organization, wherein, actual storage of data in a database was necessary. Many procedures in structural design, such as element stiffness matrix routines, generate huge amounts of data. Generally, it is unlikely that a user would want them to be stored in a database at the expense of disk space and data transportation time. The advantage of being able to query or possibly modify individual data elements does not apply to stiffness matrices, which are more or less meaningless, except to the analysis program for which each matrix has been assembled. In order to save disk space and data transfer time, it is preferable to store only those data that are required for generation of element stiffness matrix. In general a data model may be replaced by (i) an algorithm that generate the user requested information and (ii) a set of (condensed) data which will be used by the algorithm to generate the user requested information. Therefore, in an algorithmic model, data are not stored but are generated whenever they are needed. Algorithmic model provides a means for selecting a mixture of algorithms and stored data in a way that is most efficient for any given application.

Study of resource aspects must be made for deciding suitability of an algorithmic model or a data model. In cases where the items being modeled is rich in empirically derived data (for example, steel codes for allowable stress calculations) the algorithmic model uses a simple algorithm with a conventional data model. At the other extreme, where all properties of an item can be generated (for example, element stiffness matrix calculations) the algorithmic model uses a complex algorithm with very little data. A data model is preferable when storage capacity, processing costs, usage rates are high and time to transport data, rate of change of data are low. An algorithmic model is better if storage capacity, processing costs, usage rate of data are low and transport cost are high.

Methodology for constructing an algorithmic model is based on (i) design of a suitable algorithm, and (ii) design of a (condensed) data model. Design of algorithms is dependent on the standard method of computational techniques used in practice. These algorithms must be acceptable to all users of the model. Data transfer between algorithm and (condensed) data model can be provided through database management system support. Interface between algorithmic model and applications must be designed based on consideration of simplicity and ease of use. Design of (condensed) data model is dependent on the algorithm itself. If the condensed data model uses a conventional data model, then schemes for ensuring correctness of data values in the model must be provided. This is necessary because if this condensed data model is allowed to be modified arbitrarily then resulting data generated by algorithms will become meaningless. If several algorithms are in operation each using only a portion of condensed data model, then decomposition of the data model into several low level forms will enable efficient access of data values.

5. DATABASE MANAGEMENT SYSTEM FOR STRUCTURAL DESIGN - A PROPOSAL

5.1 Introductory Remarks

We need a software to use a database that has been designed based on the methodology given in the previous chapter. A software that handles requests of users to access and store data in a form compatible with data organized at various levels between physical storage level and external level is called a database management system. A database management system conceals the complex data storage details and provide a simple view of database to the users. Such a software enables structural design application programmers and interactive users to store and retrieve required data in a simple way and thereby relieving them of unnecessary burdon of managing physical storage details.

This chapter intended to propose various components of a DBMS and their functional requirements in view of implementation of a good DBMS for structural design optimization. A DBMS should have many components such as command processor, input-output processor, file operation routines, addressing and searching schemes, and memory management schemes in order to provide simple and efficient means of data storage and retrieval. Functions and requirements of these components are described in Section 5.2 with reference to structural design applications. In Section 5.3 to 5.5(a) proposal of syntactic rules (grammer) for languages used by structural designer to define data view, manipulate data and query data is given. Finally in Section 5.6, existing database management systems are reviewed and their features tabulated.

5.2 Requirements of a Database Management System

In this section requirements of a database management system for structural design optimization are given. Various components necessary in a DBMS and their functions are described. A layman's view of the workings of a DBMS are given below to show main events that take place while an application program uses a DBMS.

1. An application program calls a DBMS to define a database, relation, and attributes.

2. DBMS checks the user given definition for syntax.

3. User requests DBMS to store and retrieve data.

4. DBMS transfers data from user buffer to disk and vice versa.

5. DBMS stores data on disk in files at addresses allocated for data.

6. A system buffer of DBMS is used as an intermediate storage to avoid too many disk I/O operation for data transfer between user buffer and disk.

If a DBMS is to only perform operations as described above, then the implementation for such a DBMS would be very simple. But the requirements of structural design application programmer and the usage pattern are highly sophisticated and require a general purpose DBMS to satisfy their needs. Such

a DBMS should have components -- command languages, command processors input-output processors, addressing and searching routines, file definition and operation routines, memory management routines, integrity and rule processor, relational operators, and security and protection schemes. These are identified in Fig. 5.2.1. In the following subsections, the functions and requirements of these components are described.

5.2.1 Languages for DBMS Users

First requirement of a DBMS is to provide users with a convenient set of languages to give commands to DBMS to carry out users tasks. In general, a structural design application programmer is familiar with conventional language FORTRAN. Therefore, it is necessary to develop a DBMS so as to provide a compatible interface with host language FORTRAN. Since, this language alone does not answer user requirement to specify data types and manipulate data, additional set of sublanguages are required which are derived from FORTRAN language. Two important sublanguages are data definition language (DDL) and data manipulation language (DML). A data definition language is used to define database, relations and attributes. A data manipulation language is used to store, retrieve, modify and delete data in a database. In particular, these languages provide commands (nothing but FORTRAN call statements) to user. These commands enable a user to use database without any reference to the storage details. These two languages are described in detail in Sections 5.3 and 5.4. In addition to these languages, a query language is necessary for interactive user of a database. A query language consists of commands given by user in a character string form and does not have any reference to FORTRAN structure. A query language is useful for structural designer to inspect the contents of a database and manipulate them quickly using simple commands. The details of a query language that is suitable for structural design are given in Section 5.5.

5.2.2 Command Processor

User given commands have to be checked before they are executed by a DBMS to avoid erroneous operations on a database. Commands of DDL and DML involve subroutine call statements where as query language commands contain character strings. It is necessary to verify these commands for syntax and also against illegal use of commands in any database operations. In the case of subroutine call statements, the number of arguments, type of arguments and value of arguments have to be verified. For example, if user has requested operations on a nonexistant database, command processor must issuean illegal operation flag. Interactive user is bound to commit a large number of mistakes while issuing commands spontaneously. Therefore, a more sophisticated command processor is required to verify query commands. Such a processor has to provide functions for verifying character strings, storing the strings, separating data items from a command string, identification of integer, real and character data types in the string, finding the length of a command line, locating next item in a command line, automatic generations of new commands, and finding repetitions of previous command line.

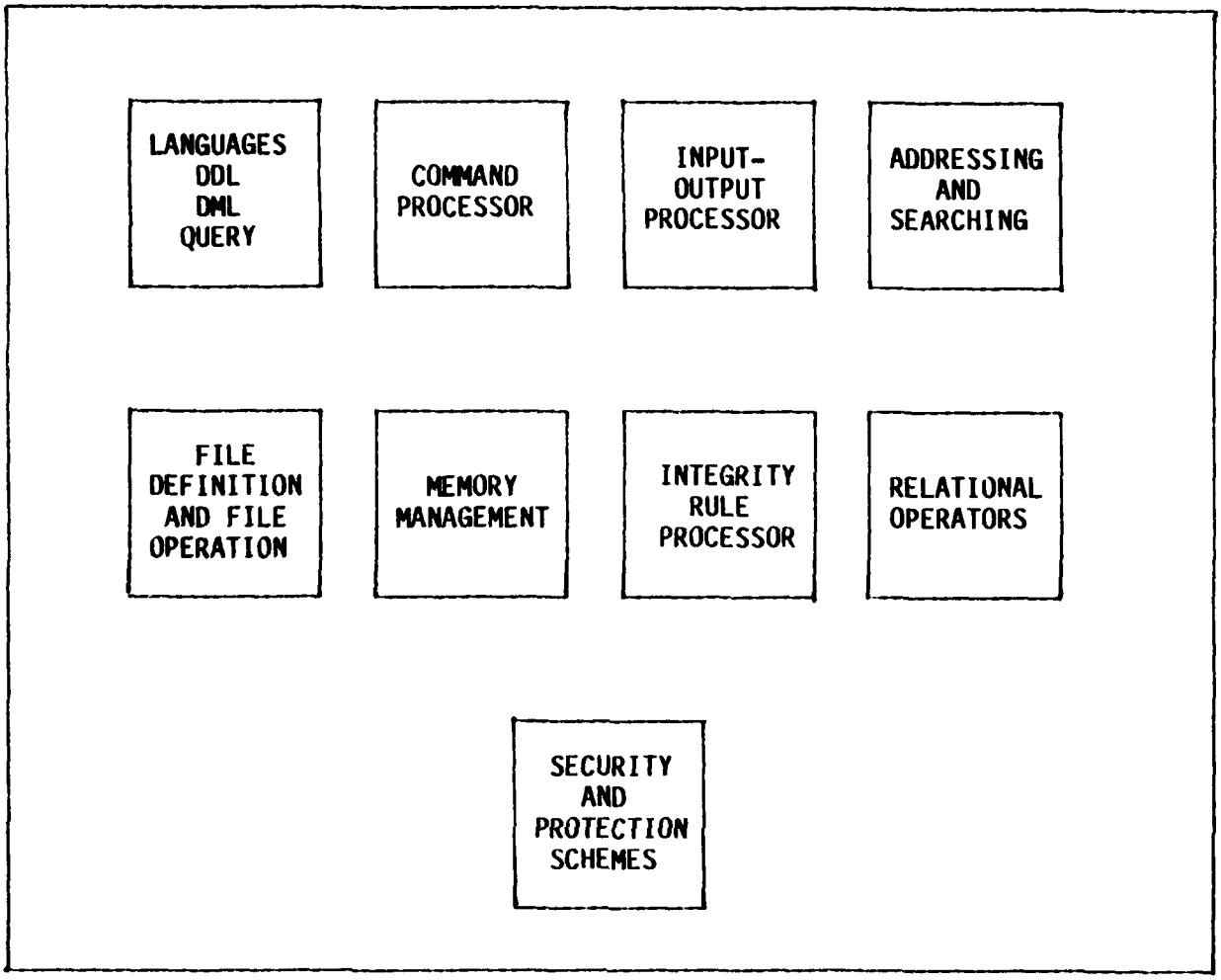


Figure 5.2.1 Components of a Database Management System

5.2.3 Input-Output Processor

Input-output processor is the basic component of a DBMS which handles all requests of the users to store, retrieve, modify and delete data. Storage and retrieval of data are done using data sets and relations as a basis. Data sets are nothing but sets of data stored in row, column or submatrix order. Relation is a two-dimensional table of data. User requests to store or retrieve portions (for instance a set of rows) of data set or relation at a time by providing them in the user buffer. Functions of I/O processor are to verify the correctness of data manipulation operations and to perform data transfer between user buffer and disk files. Existence of data sets, and relations are verified before any data is transferred between user buffer and database. If the data manipulation is based on primary keys, then I/O processor checks the existence of key values. Many structural design applications, also operates on rows of data set and relations. In such a case, I/O processor is required to verify the row numbers of data sets and relations. If data manipulation is based on certain conditions of data value in a data set or a relation (for example, modify coordinate values of nodes 5, 21, 27) then I/O processor is required to provide capability for such manipulations.

Main function of a I/O processor is to transfer data between user buffer and disk. A number of intermediate operations must be performed by I/O processor before any data transfer is made. First, data from user buffer is transferred to a system buffer of DBMS to avoid too many disk I/O. I/O processor requests memory management schemes to allocate and control space required in system buffer. Then, it requests addressing routines to provide physical storage location for placement of system buffer data when it becomes full. Also, file operation routines, integrity rule processors, relational operator and security and protection routines are called during the operation.

5.2.4 Addressing and Searching

Input-output processors cannot directly get the required data from a data set or a relation till the address to the stored data is determined. Addressing routines determine the physical storage location for a data set or a relation given a particular row number or primary key values. There are several methods to determine the addresses. Important methods that can be considered for implementation are indexing and hashing methods. When index is used for addressing, address to blocks of records are maintained in a table. A scan of address determine the block containing the required data. Then a serial search for a required row or primary key value determine its address. In hashing method, the primary key or row number is converted into a random number and is considered as the address for the data. Disadvantage of hashing method is that storage utilization in disk is low and there is some chance of clashing addresses of two data.

Index for addresses are generally large if database is big. Locating a particular index efficiently is important to save time of search. Out of several search techniques, B-tree search is popular. One index is placed at each node of a tree. Search beings at the top of a tree to locate a particular index. Depending on whether the given index is less than or greater than the index at a node, search is made toward left or right of the tree.

5.2.5 File Definition and File Operations

At the physical storage level, database consists of either a single stored file containing data or several files linked together as a unit. File definition and file operation routines are required in a DBMS. Function of file definition routine are naming of files, allocating logical unit numbers, specifying type of file access (random or sequential), specifying physical storage block size, etc.

Actual data and data definitions (name of data set, attributes, sizes) may be stored in a single file or can be separately stored in different files. File operations consists of opening, closing and deleting files. A file once opened for reading and writing should be closed at the end of file operations. File compression is done to recover unused space.

5.2.6 Memory Management

Structural design application programs use data from several data sets and relations at a particular design stage. Also, they use data of different portions of the same data set and relation in arbitrary sequence. In such situations, we are faced with the problem of accommodating the required data in the primary memory and at the same time reduce I/O activity to lessen computation time. Efficient use of primary memory is possible through judicious allocation of available space. Memory management scheme dynamically controls the available memory space. The memory is organized into pages (which is a unit of transfer between the database and primary storage) and page sizes are assigned. The size of the page is set to multiple of a physical record. Larger the page size, better the performance as less I/O is needed to get the required data. However, space may be wasted if there are too many partially filled pages. Small page size leads to increase in page replacement activity and maintenance of large page size table. Variable length pages require tedious programming effort.

Another important function of memory management scheme is page replacement activity. Memory management scheme should keep count of the pages in the memory. Paging scheme may adopt some page replacement algorithm. Least recently used (LRU) page replacement is the most commonly used method. In LRU algorithm, a page counter is maintained for each page and updated each time the page is used. When a page is to be replaced, the page with the highest counter value becomes the candidate. A page replacement is done when no free pages are available. A page not modified is over written instead of replacement.

Memory management scheme should be developed such that user has some control over the size of pages. This feature helps in determining the appropriate page size while operating on larger matrices of finite element analysis and design optimization problems. Fragmentation of large matrices on pages can be avoided by using page size in multiples of matrix size. Matrix operation algorithms for addition, multiplication and transpose by row operations can use page size in multiples of number of rows of a matrix. The algorithms that solve large order simultaneous equations by submatrix approach require at least a few submatrices to be present simultaneously in the memory. In such a case, allocation of one submatrix per page induces fewer

page faults. This leads to reduction in I/O activity of iterative algorithms and brings down the execution time.

5.2.7 Integrity Rule Processor

In Section 3.5, we observed that integrity maintenance is an important task in structural design database. Integrity rules are enforced in practice by providing key constraints, referential constraints and other constraints. It is necessary to provide facilities in a DBMS to enforce these integrity constraints. Such a facility is possible through integrity rule processor. Functions of rule processor are to define constraints, check constraints during data storage and data manipulation and issue user messages in case of violation of rules. Key constraints are imposed by the processor by assigning various attributes to form key attributes. Referential constraints are imposed by specifying attributes belonging to two or more relations to share common data values. Other constraints, say for example coordinate values should be greater than zero are imposed by specifying range of values an attribute must take. If a large number of constraints are imposed, then time required to check the data during data manipulation are high. Therefore, DBMS should provide facility for the user either to check the constraints or allow data manipulation without checks. In the later option, user is responsible for ensuring validity of data.

5.2.8 Relational Operators

Database proposed in the previous chapter, is based on relational data model. Therefore, a DBMS which operates on a relational database must have relational operators to manipulate the database. Relational operators manipulate data in terms of entire sets or relations and not in terms of individual rows or columns at a time. Three types of relational operators are SELECT, PROJECT and JOIN. Each of these operators take either one or two relations as its operand and produces a new relation as its result. The SELECT operator constructs (or lists out) a new relation by taking horizontal subset (specific rows) of a relation. The rows that satisfy some condition are selected. The PROJECT operator forms a vertical subset of an existing relation by extracting specified columns and removing any redundant duplicate rows in the set of columns extracted. The JOIN operator, joins two relations, each having common column (attribute), and produces a new wider relation in which each row is formed by concatenating two rows, one from each of the original relation, such that two rows have the same value in those two columns.

5.2.9 Security and Protection Schemes

Structural design database is used by a number of application programs and users. It is necessary to ensure that database contents are not destroyed or manipulated by unauthorized users. Therefore, DBMS must have special scheme to ensure security and protection of a database. Security of a database against unauthorized use can be provided by allocating password schemes to access contents of a database. Users of database are provided with read and modify passwords to use the database accordingly. These passwords

can be assigned both at database level and individual dataset or relation level. Protection of a database is necessary to guard against destruction of a database due to computer malfunction. Periodical backup of the database will ensure such safety.

5.3 Data Definition Language

Data definition language (DDL) is a means to declare data types and logical relations among them. DDL can be used to define both external and internal views of data. External views are declared either through an application program or interactively. For the application programmer, DDL is a conventional language like FORTRAN; for interactive user it is the query language. Data definition facility at the internal level is provided by special commands (internal DDL) built into a DBMS. Note that external view of users is only a portion of internal view of data in a database. It is necessary to build both external data definition and internal data definition language such that they do not contain reference to physical storage details on disk. Thus, any change in storage structure of data on disk will not force modification of application programs.

Data definition language for the application programmers consist of those declarative constructs of FORTRAN needed to declare database objects: Variables and arrays, FORTRAN data types, extens ons to FORTRAN to support objects not handled by FORTRAN. External views of data is defined by application programmers using relational and data set constructs. Since FORTRAN does not provide facility to express relations and data sets, we need to provide extensions to FORTRAN to define data. Such an extension is possible through FORTRAN CALL statements and is provided as a part of DBMS. Arguments of the subroutine call statements, for example, specify database name, relation name, attribute name and size, etc.

Development of a DDL for structural design application should be based on several considerations. One of the major consideration should be to keep syntax of DDL concise and to be easily understood by application programmers. Furthermore, since, the DDL is used in structural design computing, all the data types of FORTRAN must be allowed -- integer, real, double precision and character. Data elements allowed by the DDL also include those of FORTRAN scalars and arrays. DDL should support both relations and data set definition. Relation definition require specification of relation names, attributes (names, type, and size), key attributes, and variable length attributes. Data set definition requires specification of data set name, data set type, row size, column size and/or submatrix size. In addition to relation and data sets, DDL should be able to define numerical data. Large order matrices are organized in banded, hyper matrix or skyline form. Special facilities must be provided to define these large order matrices. There are many instances in structural design data where maximum size of data is known. For example size of stiffness matrix is known in advance. In such a case, provision in DDL for specification of maximum number of data occurrence will enable DBMS to conserve storage space and provide maximum efficiency. Another feature that should be included in DDL is data redefinition capability. This can be accomplished by providing several transformation of the same data to different application programs. It may be convenient, for instance, to represent a matrix in two different ways in different external

views. In one view the matrix might be defined as two-dimensional array and in another view it might be defined as a vector. One important feature that should be incorporated in DDL of structural design application is compilation independent data definition facility. This means that DDL must have facility to define data types and relationships during run time. This feature is necessary because data definition in many instances are not known till certain stage of processing is complete. An example of this is number of degrees of freedom which is not known to define size of assembled stiffness matrix, till input data is processed. Finally, consideration for development of DDL should include database security. Mechanisms for security such as read and modify password must be provided at both relation and data set level.

Internal DDL could be again an extension to the programming language or special commands supported by a DBMS. Considerations for developing an internal DDL are same as described above.

A proposal of DDL for structural design application is given in the Appendix 2. The data definition language has following features:

1. Database definition with provisions to define global and local databases. Each database can be either permanent or temporary. Temporary meaning that database is deleted at the end of execution.

2. Database user identification provision. Database can be used either by a owner or a user. Database access is defined either by read or modify rights.

3. Data set definition facility with integer, real and double precision data types. Scalars, vectors and matrices can be defined.

4. Relation with attributes of interger, real and double precision data types. Attributes can be scalars, vectors, and matrices. Provision for key attributes and variable length attributes are made.

5. Termination of data set definition and data redefinition facility is available.

6. Numerical data definition with provision to define square, triangular, banded, hyper and skyline matrix is provided.

DDL syntax of Appendix 2 uses Backus-Naur Form as a tool for defining syntax of languages. BNF is a common tool for formally describing a programming language syntax. A syntax is nothing but grammar of a language.

5.4 Data Manipulation Language

Data manipulation language (DML) is used to store, retrieve, modify and delete data in a database. For application programmers DML is provided through subroutine call statement; for interactive user it is provided through query language. At the internal level, store command of DML is generally sufficient to fill the database with values. Such a command is usually provided as a special command built into a DBMS.

Development of a DML for structural design application should be based on several considerations. The DML commands should be simple as they are frequently used in an application program. The DML should not contain any reference to the storage structure details. DML should have capability to access data from different databases. Relation and data sets are frequently accessed and stored row-wise. Therefore, commands in DML must facilitate this operation. Also, it should be possible to store and retrieve rows in a sequential order or in a random order. Further, each row can contain data from a set of attributes each belonging to different data type -- integer, real, double precision. Therefore, commands should be designed to accommodate multiple data types for data manipulation operations. Many structural design application programs require data of a particular attribute of a relation. For example, some programs need only degree of freedom attributes in a node-coordinate-degree of freedom relation. Consideration in design of DML should include data manipulation in terms of single attributes or a set of attributes. Further, it should be possible to select any required values from an attribute that satisfies certain conditions. Special commands must be provided in DML to specify conditions on data values that are to be manipulated.

Further consideration in the design of DML is based on matrix manipulation requirements. Large order matrices are generally stored and retrieved row-wise, column-wise and submatrix-wise. Data manipulation commands must include operators to manipulate matrix data in terms of rows, columns and submatrix order. Matrix operations such as transpose, multiplication require column-wise retrieval of data stored in row-wise pattern. Also, DML must be able to retrieval matrix data in various external view such as banded, skyline as depicted in Fig. 4.5.6. Special provisions should be made in the commands to indicate null rows (zero elements in a row), null columns and null submatrices.

There are a number of other considerations for designing DML. Commands should include utility and data definition list facility. Utility commands of DML are open database, close database and error display commands. Commands should be able to provide facility for opening and closing of a number of databases at various stages of execution. Data definition list command will enable application programmer to check and use detailed definition of data objects stored in the database. Error display commands facilitates application programmer to investigate the cause of errors.

A proposal of DML for structural design application is given in Appendix 3. The data manipulation language has the following facilities:

1. Open and close command to open and close a database.

2. Retrieval command. This has facility to retrieve data set and relation.

3. Store command to fill data set and relations with values.

4. Delete command to delete parts of data sets and relations.

5. Modify command to modify parts of a data set or a relation.

6. Remove command to eliminate a data set or a relation.

7. Copy command to copy data from one data set or relation to another.

8. Store, retrieve, modify and delete commands for matrix data.

9. Rule command to specify conditions on data values that have to be retrieved.

DML syntax of Appendix 3 is given in Backus-Naur Form. Description of BNF are same as those given for DDL.

5.5 Query Language

Structural designers often want to use a database interactively to find out various parameters of design and to modify them by using simple commands. Query language provides a simple set of commands to interactively define and manipulate data in a database. It can be used by any nonprogramming user and does not require knowledge of high level languages like FORTRAN. Data definition of query language includes database, relation and attribute definition, rule specification, and authorization procedures. Data manipulation of query language includes store, retrieve, modify and delete commands. In addition to these commands relational operators like SELECT, PROJECT and MODIFY are provided.

A query language should satisfy several requirements. These are data independence, simplicity, nonprocedurality, extendability and completeness. Data independence refers to independence of logical data structure and storage structure definition. Query language should not contain any reference to storage structure. Nonprocedurality means that user should be allowed to give commands in any arbitrary order and should not restrict them to follow procedures to query a database. Query language should be easily extendable to incorporate any special function into it. Query language should be complete in the sense that it possesses all the required commands to query all types of data in the database. Query commands must be general and not limited to any special case. Query of large order matrices requires special features in query language so that data can be displayed in parts.

General syntax of a query language is:

<command> <Expressions clause> <conditional clause>

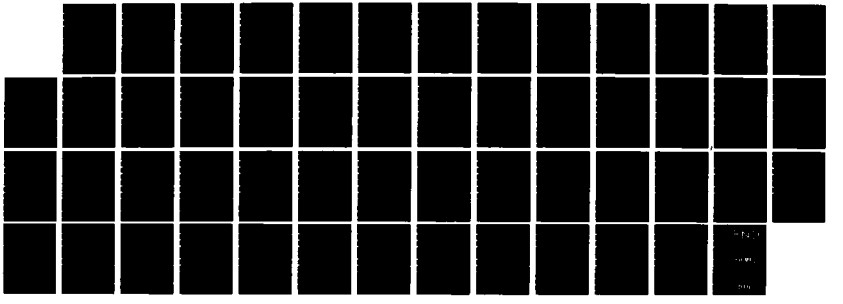
The command is a name interpreted by a DBMS to execute certain procedure for defining and manipulating data. Some typical commands required for structural design are SELECT, LIST, CHANGE, RENAME, OPEN, CLOSE, LOAD, DEFINE, and EXIT. The second component is expression clause which is a group of words specifying names of relation and attributes. The third component, conditional clause, allows a user to specify a condition on the data for which command is executed. The conditions may be, for example, GT.100; LT.20, ROWS.EQ.10.

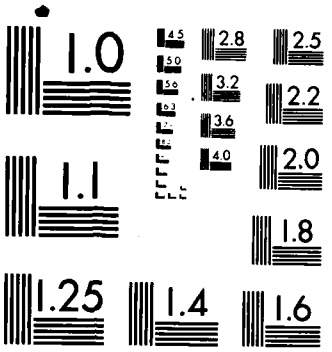
5.6 A Review of Database Management Systems

In this section, a review of existing database management systems is made. This review enables us to evaluate, select and modify a database

AD-A162 101 DATABASE DESIGN METHODOLOGY AND DATABASE MANAGEMENT SYSTEM FOR COMPUTER-A (U) IOWA UNIV IOWA CITY APPLIED-OPTIMAL DESIGN LAB T S MURTHY ET AL DEC 84 2/2

UNCLASSIFIED CAD-SS-84-20 AFOSR-TR-85-1071 AFOSR-82-0322 F/G 5/1 NL





management system so that it is suitable for structural analysis and design purpose. The following fifteen database management systems are reviewed. The capability of these systems are emphasized and important features are tabulated.

DELIGHT - Design Language with Interactive graphics and a Happier Tommorow
DATHAN - A data handling program for finite element Analysis
EDIPAS - An Engineering Data Management System For CAD
FILES - Automated Engineering Data Management System
GIFTS - GIFTS Data Management System
GLIDE - GLIDE Language with Interactive graphics
ICES - Integrated Civil Engineering System
IPIP - Information Processor for IPAD
PHIDAS - A Database Management System for CAD
REGENT - A System for CAD
RIM - Relational Information Management System
SDMS - A Scientific Data Management System
SPAR - SPAR database management system
TORNADO - A DBMS for CAD/CAM system
XIO - A Fortran Direct Acess Data Management System

DELIGHT. It stands for Design Language with Interactive Graphics and a Happier Tommorow (Nye, 1981). In its philosophy, the DELIGHT system is very close to the GLIDE system (Eastman and Henrion, 1980). DELIGHT is an interacive programming language. It has good extension and debugging capability. It provides high-level graphic commands, a built-in editor and a well-defined interface routines. A single statement, procedure or part of an algorithm can be tested without having to write and load/link a program. The system relies on virtual memory management of the operating system. It is difficult to use the system with large scale programs. Multiple users are not allowed in the system.

DATHAN. It stands for data handling program. It is written mainly for finite element analysis applications (Sreekanta Murthy and Arora, 1983). The program has some basic in core buffer management scheme. It has capability to store permanent and temporary data sets. Substructure files can be arranged quite easily with same data set names for different substructures. Both integer and real data types can be handled. Drawback of the system is that the user has to keep track of the location from which a new data set has to begin. The system has FORTRAN data manipulation commands which are simple to use.

EDIPAS. It stands for Engineering Data Interactive Presentation and Analysis System, (Heerema, and van Hedel, 1983). It is a tool for data management, analysis, and presentation. The data management part provides a utility to initalize a project database, input programs to load data from files into database under user controls, and a set of routines to extract data from and load data into database in a controlled way. EDIPAS allows users to name a database, a data structure, and data entities. It allows user to employ one or more hierarchical levels. The data is stored in entities called blocks. A data block allows matrices, single values and characteristic values as data elements. A database administration support provides initialization of database, access to users, deletion of data structures, audit database

contents, and back-up facility. The system does not have data redefinition facility. Improvements are being done to include redefinition facility in order that the data structures and their levels can be manipulated. Extension of authorization provision from database level to the level of data element is being incorporated.

FILES. It is an automated engineering data management system (Lopez, 1974). It is extremely flexible with respect to the definition of a database and methods of accessing it. Information storage and retrieval may be performed using problem-oriented languages. Hierarchical data structure is provided. For example matrix type of data encountered in finite element application can be organized using hierarchical data structure. The first two levels in hierarchy may contain pointers to the third level containing actual matrix data. The program allows dynamic memory allocation. Data transfer takes place between FORTRAN common block and database. FILES has a data definition language. The system does not have data mapping language to specify mapping of data items and arrays to an external device. The data definition language (DDL) depends on the problem oriented language (POL). Therefore DDL cannot be used independently. The system requires a distinct data management compiler.

GIFTS. It is an interactive program for finite element analysis (Kamel, McCabe and Spector, 1979). It is a collection of modules in a program library. Individual modules run independently and communicate via the unified database. The database manager processes requests for opening a file, closing a file, storing data set in a file, and retrieving data set from a file. The program has memory management scheme. Each data set is stored in a separate random access file. Paging is carried out within the working storage. A unique set of four routines is associated with a data set for opening and initializing the working storage, for reading a data set, for creating/modifying the data set, and for realizing the working storage. Drawbacks of the system is that every new data set requires created four new routines to be written. Each data set is associated with a separate common block, thereby increasing the number of common blocks in the system. The data manager is application dependent and cannot be used as a stand alone system.

GLIDE. It is a context-free database management system (Eastman and Henrion, 1980). It is designed to provide a high level facility for developing individualized CAD system. It can be viewed as a language, a database management system, and a geometric modelling system. It allows users to define new record types known as FORM that consist of a set of attribute field. It provides primitive data type set to organize a database. It provides excellent geometric modelling system or a graphic system. Drawback of GLIDE is that it does not allow multi-dimensional arrays.

ICES. Integrated Civil Engineering System is a computer system designed for solving civil engineering problems (Roos, 1966). ICES consists of a series of subsystems each corresponding to an engineering discipline. It provides a Problem Oriented Language which can be used to write subsystem programs (e.g., coordinate geometry program, stress analysis program). Command Definition Language is used by a programmer to specify the structure and required processing for each subsystem commands. A Data Definition Language is used to specify the subsytem data structure. It uses its own programming language called ICETRAN (ICES FORTRAN) and has a precompiler which translates ICESTRAN to FORTRAN statements.

Dynamic data structuring capability is provided in the system which helps to organize dynamic arrays in the primary memory. Hierarchical data structure is used for data modelling. Three hierarchical levels: equivalence class, members, and attributes are provided. Data is stored on secondary storage using random access files. Data management program uses buffers to convert logical records to physical records. Identifier is supplied by the programmer which is a pointer giving the position on secondary storage of physical record. The programmer has a choice to store data using dynamic arrays or using data management system depending on amount and use of the data. Drawback of the system is that it uses precompiler ICETRAN to convert to FORTRAN program instead of directly to machine language. Physical storage of data requires knowledge of address and pointers which the programmers have to give. Only three levels of hierarchy is adopted and it is difficult to extend to many levels of hierarchy.

IPIP. It is a state-of-the-art database management system satisfying engineering requirements (Johnson, Comfort and Shull, 1980). It offers a number of capabilities ranging from support for multiple schemas and data models to support for distributed processing, and data support for distributed processing, and data inventory management. An integrated software architecture supports all user interfaces: programming languages, interactive data manipulation and schema languages. IPIP supports a multiple-schema architecture of ANSI/SPARC database group. Three types of schemas -- conceptual, external and internal schemas are supported. IPIP schema and data manipulation languages exhibit a high degree of integration and compatibility. The logical schema supports both the network and relational data models, and, functionally, the hierarchical data model. The internal schema of IPIP is written using the internal schema language compiler. The internal schema language overlaps that of the logical schema language to the greatest practical extent to minimize the amount of schema language with which the administrator must deal. IPIP software subcomponents consists of user interface, and data manager. Software of user interface is made of precompilers, query processor and compilers. Data manager software is made of scheduler, message procedure interface processor, common semantic processor, database control subsystem, data manipulation subsystem, record translator, presentation service, access module, resource manager and stubs.

PHIDAS. It is a data management system specially designed for handling a collection of structured data on minicomputers (Fischer, 1979). The architecture of PHIDAS is in accordance with the ANSI-3 schema. It has an external subschema based on network model of CODASYL and an internal schema for physical tuning particularly suited for engineering database. The data description language is provided to describe schema and sub-schema. PHIDAS also has a storage structure description language. Data manipulation language is FORTRAN call statements to subroutines. Drawback of the system is that it is difficult to represent matrix type data.

RIM. It stands for Relational Information Management system (Comfort, Erickson 1978). RIM has capability to create and modify data element definition and relationships without recompiling the schemes or reloading the database. RIM provides capability to define new types of data for use in special application such as graphics. RIM supports three types of data: real, integer, and text. Data definition and data manipulation languages are available to define or manipulate relations. The user has capability to

project, intersect, join and subtract relations. RIM has good query language. RIM's modification commands permit the user to update relation definition, change data values, attribute names, delete tuples and delete the entire relation. Utility commands such as LOAD, and EXIT are provided to load a new database and close an existing database. Drawback of RIM is that it does not allow relation having row size more than 1024 computer words. The application oriented FORTRAN call statements do not have capability to define attributes, relations, rules, etc., required in defining a schema. The system does not support management of a temporary database. Simultaneous operations on a number of databases is not possible.

REGENT. It is a system for the support of computer-aided design (Leinemann and Schlechtendahl, 1976). The main goal of the development was to provide a so-called "system nucleus" in the sense of ICES. Improvement claimed for the system is that it has a powerful base language PL/1 instead of FORTRAN. Interactive use has been considered in system development. The database management of REGENT provides facilities to compress a database, copy data between databases, and to change name and size of data elements. The database of REGENT is not a database in the usual sense. It is some sort of partitioned data set concept, built up using a tree structure of sequential files, but the internal structure of these files is known only to those programs that use them.

SDMS. It is a database management system developed specifically to support scientific programming applications (Massena, 1978). It consists of a data definition program to define the form of databases, and FORTRAN compatible subroutines to create and access data within them. Database contains one or more data sets. A data set has form of a relation. Each column of a data set is defined to be either a key or data element. Key must be a scalar. Data elements may be vectors or matrices. The element in each row of the relation forms an element set. Temporary database capability that vanishes at the end of a job is provided. A scientific data definition language provides a program-independent data structure. Both random and sequential access of data set is possible. Data elements include scalars, fixed and variable length vectors, fixed and variable-size matrices. Data element types include text, real and integer. Drawback of the system is that it does not have a query language. Generalized database load/unload is not available. Double precision data type is not allowed. The system is implemented only on Cyber series computers.

SPAR. The computer program is a collection of processors that perform particular steps in finite element analysis procedure (Whetstone, 1977). The data generated by each processor is stored on a database compiler that resides on an auxillary storage device. Each processor has a working storage area that contains the input and the computed data from the processor. Allocation of spaces in the storage area is a problem dependent and is dynamically allocated during execution. Data transfer takes place directly between a specified location on disk using a set of data handling utilities. SPAR database complex is composed of 26 data libraries or data files. Libraries 1 to 20 are available for general use. Libraries 21 to 26 are reserved for temporary and internal use. The database manager uses a master directory to locate the table of contents which in turn is used to locate the data sets in the database. Physically, the auxillary storage is divided into sectors of fixed size and each read/write operation begins at the beginning of a

sector. Drawback of the system is that it does not provide either hierarchical or relational data structure. Excessive fragmentation may take place if the sector size does not happen to be an integral multiple of the data that is stored.

TORNADO. It is a DBMS system developed for CAD/CAM application (Ulfsby, Steiner and Oian, 1979). It is a CODASYL network system written in FORTRAN and is very useful for handling complex data structures. It handles variable object length and dynamic length records. System allows different data types - integer, real, character, double precision, double integer, complex and logical data. The system has easy to use data definition language and data manipulation language. TORNADO system is highly portable. Data in the database can be accessed by name. There is no restriction on data set types and allows many-to-many relationships. Drawback of the system is that the size of a data object defined by the system is limited by the largest integer value that can be represented in the computer. The size of the database is limited by the maximum size of a file. A multi-file version is not available. The database cannot be used by multiple users at the same time.

XIO. It is a set of subroutines that provides generalized data management capability for FORTRAN programs using a direct acces file (Ronald, 1978). The system allows arrays of integer, real double precision and character data storage. Both random access and sequential access of data is provided. Variable length record I/O is allowed in the system. Bit map scheme is used to identify the unused space for storage of data to minimize disk storage requirement. The program allows restart facility using saved file following completion of a partial execution or after a program termination. The system at present is only implemented on IBM360 or DEC PDP11 computing systems. The system does not provide data definition language. It does not provide either hierarchical or relational data structures.

Capabilities of various systems are summarized in the table.

TABLE 5.3.1 FEATURES OF VARIOUS DATABASE MANAGEMENT SYSTEMS FOR ENGINEERING APPLICATIONS

A = Available
AS = Available but Specialized
H = Hierarchical
F = FORTRAN
N = Network
PL = PL/1
R = Relational
O = Others
D = Data set

NO	DBMS	HOST LANGUAGE	EXTERNAL MODEL	INTERNAL MODEL	DATA DYNAMIC DATA DEFINITION	DATA DEFINITION LANGUAGE	DATA MANIPULATION LANGUAGE	MEMORY MANAGEMENT	APPLICATION INTERFACE	INTERACTIVE CAPABILITY	MATRIX DATA ORGANIZATION CABABILITY	GEOMETRIC DATA CAPABILITY	SIMULTANEOUS ACCESS TO MULTIPLE DATABASE	MULTIPLE USER FACILITY	TEMPORARY DATABASE
1	DELIGHT					AS	AS	A	A	A	A	A			
2	DATHAN	F		D	A	A	A	A	A		AS				A
3	EDIPAS		H			A	A		A	A	AS	A			
4	FILES	F	H			AS	A	AS	A		A		A		A
5	GIFTS	F		D			A		A		AS	AS			A
6	GLIDE					AS	AS		A	A		A			
7	ICES	O	H,D		AS	AS		AS	AS						
8	IPIP		H,N,R	A	A	A	A		A	A		A	A	A	
9	PHIDAS	F	N	A		A	A		A	AS		A			
10	RIM	F		R		A	A	A	A	A	A	A		A	
11	REGENT	PL			AS	AS		AS	AS						
12	SDMS	F		R		A	A		A		A		A		A
13	SPAR	F		D		A	A	AS	A		A		A		
14	TORNADO	F	N		AS	A	A		A			A			
15	XIO	F					A	AS			A				

6. IMPLEMENTATION OF A DATABASE MANAGEMENT SYSTEM -- MIDAS

6.1 INTRODUCTORY REMARKS

A database management system - MIDAS* has been implemented for finite element analysis and structural design optimization applications. The MIDAS implementation is based on the requirements of a database management system given in Chapter 5. MIDAS has two subsystems - MIDAS/R and MIDAS/N. These subsystems are capable of organizing data of relational and numerical models, respectively. The system has been installed on PRIME computer system. It relieves the burden of managing data for application programmers by providing user-friendly application commands. The system has sophisticated interactive commands to query the database. The MIDAS system can be used either interactively or through application programs. The implementation details of MIDAS/R and MIDAS/N are given in Sections 6.2 and 6.3, respectively.

6.2 IMPLEMENTATION OF MIDAS/R

In this section, capabilities, database organization, data definition, data manipulation and query commands of MIDAS/R are described. Also, details of the system architecture are given. MIDAS/R database management system is based on relational data model. The system is developed by modifying and extending the RIM program (Relational Information Management System).

6.2.1 Capabilities of MIDAS/R

MIDAS/R is written in FORTRAN-77. The system does not have any machine dependent instructions and therefore it is highly portable. It has data definition and data manipulation commands which are simple to use for application programmers. It has sophisticated interactive commands to query the database, modify the database and display the database schema. The system has capability to store, retrieve, modify and delete a database using both application call statements and interactive commands. The data can be integer, real, double-precision and character words. The data can be organized in the form of relations. The system has powerful relational algebra commands like SELECT, PROJECT and JOIN. MIDAS/R has capability to provide access to the database simultaneously for multiple users. Database security is provided through two level password system. Error recovery mechanisms are available.

6.2.2 Database of MIDAS/R

MIDAS/R has capability to create a number of databases. The databases can be used one at a time. The database can be either permanent or temporary. The size of a database is unlimited but depends only on the availability of disk space. Database of MIDAS/R can hold any number of relations each of which is identified by a unique name. A relation can store data of a number of attributes. An attribute value can be a single data item, a vector or a matrix. Variable length rows of a relation can be stored.

*(Management of Information for Design and Analysis of Systems)

6.2.3 Data Definition Commands of MIDAS/R

Data definition commands are used in an application program to define a database, relations and attributes. These commands are FORTRAN subroutine call statements. These commands were not available in RIM program. The data definition commands of MIDAS/R are described in the following paragraphs.

Database Initialization:

```
CALL RDBINT
```

This command initializes MIDAS/R. Before using any other commands of the system, this command must be used.

Database Definition:

```
CALL RDBDFN (NAME, STAT, IERR)
```

NAME = Name of the database
STAT = Permanent or temporary status of the database
IERR = Error Code

A unique database can be defined using this command. A temporary database is deleted when it is closed.

Relation Definition:

```
CALL RELDFN (NAME, RNAME, NCOL, CNAME, CTYPE, IELM, JELM, KEY, IERR)
```

NAME = Name of the database
RNAME = Relation name
NCOL = Number of attribute columns
CNAME = A vector of attribute names
CTYPE = A vector of attribute type
IELM = A vector of row size of attributes
JELM = A vector of column size of attributes
KEY = A vector of key attribute indicator
IERR = Error code

A relation can be defined using this command. Relation name and attribute names must be unique in a database. A row and column intersection in a relation table can contain either a single data item, a vector or a matrix. Details of data type and layout of data in a typical relation are given in Figs. 6.2.1 and 6.2.2, respectively.

Data Set Definition:

```
CALL RDSDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)
```

NAME = Name of the database
DSNAME = Name of the data set

Description	TYPE	IELM	JELM
Integer	INT	1	1
Real	REAL	1	1
Double Precision	DOUB	1	1
Integer Vector	IVEC	n	1
Real Vector	RVEC	n	1
Double Precision Vector	DVEC	n	1
Integer Matrix	IMAT	m	n
Real Matrix	RMAT	m	n
Double Precision Matrix	DMAT	m	n
Text or Character	TEXT	n	1

NOTE: Values of IELM and JELM if 0 indicate that data is of variable length.

Figure 6.2.1 Data Type and Size of a Relation

Attribute A Key	Attribute B Type INT	Attribute C IVEC	Attribute D IMAT
1	x	x x x x	x x x x
			x x x x
			x x x x
			x x x x
2	x	x x x x	x x x x
			x x x x
			x x x x
			x x x x
..	..	..	. . . . .
r	x	x x x x	x x x x
			x x x x
			x x x x
			x x x x

NOTE: For Attribute A IELM = 1, JELM = 1
Attribute B IELM = 1, JELM = 4
Attribute C IELM = 4, JELM = 4

Figure 6.2.2 Layout of Data in a Typical Relation

DTYPE = Data type (see Fig. 6.2.1)
IELM = Row size of a attribute (see Fig. 6.2.1)
JELM = Column size of a attribute (see Fig. 6.2.1)
IERR = Error code

A data set is defined as a collection of data belonging to same data type such as single data item, vector, or matrix. In a sense, a data set is a relation having only one attribute. Name of data set has to be unique in a database.

Data Set Redefinition:

CALL RDRDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)

Arguments are same as in data set definition. This command redefines a data set using new data type, and new attribute size. Old data set definition and its data is lost.

Data Definition Ending:

CALL RDSEND (IERR)

IERR = Error Code

After database, relations and data sets have been defined, data definition process is ended by calling this routine. During execution of this call statement, the data definition is verified and compiled internally.

6.2.4 Data Manipulation Commands MIDAS/R

Data manipulation commands open, close, store, retrieve, modify, and delete data, rename a relation or a data set, rename an attribute and copy data sets in a database. These commands were not available in the RIM program. The function and description of these commands are given in the following paragraphs.

Open a Database:

CALL RDBOPN (NAME, STAT, IERR)

NAME = Name of the database
STAT = Permanent or temporary status of the database
IERR = Error code

A database closed earlier can be opened using this command. A database has to be opened before any operation on the database is performed.

Close a Database:

```
CALL RDBEND (IERR)

    IERR   = Error code
```

A database is closed using this command. Execution of this command transfers the system buffer data into the database and closes the file.

Store Data in a Relation:

```
CALL RDSPUT (NAME, DSNAME, KROW, UBUF, IERR)

    NAME   = Name of the database
    DSNAME = Name of a relation
    KROW   = Row number
    UBUF   = User buffer which contain data
    IERR   = Error code
```

Data can be stored into a relation from application program work area (user buffer) by using this command. Data is transfered from user buffer to the specified row of a relation. If more rows have to be stored, a FORTRAN DO loop over the row number in the application program will transfer all the required rows. More details of this command are given in the user's manual (Sreekanta Murthy and Arora, 1984).

Retrieve Data from a Relation:

```
CALL RDSGET (NAME, DSNAME, KROW, UBUF, IERR)
```

Data can be retrieved from a relation into a user buffer using this command. Requested row of a relation is transfered from a relation into user buffer. FORTRAN DO loop over the row number is necessary if more than one row has to be retrieved. Data can be retrieved in the same order as it was stored by initializing row number as zero. Data of a relation satisfing certain condition (for example, attributes having certain values) can be retrieved into user buffer. User specifies the condition on data values that must be satisfied for retrieval, by using RDSRUL command (explained later). The details for various ways of data retrieval is given in the user's manual (Sreekanta Murthy and Arora, 1984). Arguments are the same as in RDSPUT.

Modify Data in a Relation:

```
CALL RDSMOD (NAME, DSNAME, KROW, UBUF, IERR)
```

Once a database is loaded using RDSPUT command, it can be modified by calling RDSMOD. This routine modifies a row of the relation. RDSGET routine is called before calling this subroutine. A row of a relation is retrieved into user buffer and this row or a part of the row can be modified and stored back using the command. Arguments are the same as in RDSPUT.

Delete Rows of a Relation:

CALL RRWDEL (NAME, DSNAME, KROW, IERR)

Rows of a relation can be deleted using this command. This is useful in eliminating unwanted values in a relation. Arguments are the same as in RDSPUT.

Delete a Relation:

CALL RDSDEL (NAME, DSNAME, IERR)

This command deletes a relation from a database. Arguments are the same as in RDSPUT.

Rename a Relation:

CALL RDRNAM (NAME, OLDNAM, NEWNAM, IERR)

NAME = Name of a database
OLDNAM = Old name of the relation
NEWNAM = New name of the relation
IERR = Error code

An existing relation's name can be changed to a new name using the command.

Rename an Attribute:

CALL RRNATT (NAME, DSNAME, OLDATT, NEWATT, IERR)

NAME = Name of a database
DSNAME = Relation name
OLDATT = Old attribute name
NEWATT = New attribute name
IERR = Error code

Copy a Relation:

CALL RDSCPY (NAME1, NAME2, DSNAM1, DSNAM2, IERR)*#AM2, IERR)

NAME1 = Name of a database containing data
NAME2 = Name of a database where data has to be copied
DSNAM1 = Relation name containing data
DSNAM2 = Relation name to where data has to be copied
IERR = Error code

Using this command, data from one relation can be copied to another relation. Both the database and relation must have been defined before copying the data.

Condition Specification for Retrieval of Data:

```
CALL RDSRUL (NUM, ATNAM, COND, VALUE, BOOL, IERR)

    NUM    = Number of conditions
    ATNAM  = A vector of attribute names
    COND   = A vector of logical operator (EQ, GT, LT)
    VALUE  = A vector of attribute values
    BOOL   = A vector of Boolean operator (AND,OR)
    IERR   = Error code
```

As mentioned in RDSGET command, data values satisfying certain conditions can be retrieved. The conditions can be specified using RDSRUL command. This command must be executed before calling RDSGET routine. A maximum of ten conditions can be specified at a time. The following example, illustrates use of this command.

Condition on a relation X:

```
Attribute A•GT•15•3 •AND• Attribute B•LT•20•1
Use NUM = 2; ATNAM(1) = 'A'; ATNAM(2) = 'B'
    COND(1) = 'GT'; COND(2) = 'LT';
    VALUE(1) = 15.3; VALUE(2) = 20.1;
    BOOL(1) = 'AND'
```

6.2.5 Interactive Commands

MIDAS/R provides interactive support for creating, updating, modifying, and deleting a database. Interactive commands are general and can be used in any application. The system provides terminal prompts for the users to respond with appropriate commands. The interactive session starts with a display of MENU and requests the user to choose one of the five options: CREATE, UPDATE, QUERY, COMMAND and EXIT. The interactive session ends with an EXIT command. The detailed commands for each of these options are entered at appropriate instant. They are given in the following paragraphs.

Database Definition:

```
CREATE XXXX
```

Create command branches out to interactively define a new database. System responds by requesting database, relation and attribute names, and authorization access details. User can supply these data at the prompt. Details of using this command are given in user's manual (Sreekanta Murthy and Arora, 1984).

Loading a Database:

After a database has been successfully created, it may be loaded before ending 'create' session. for user response 'Y' for the load prompt, the list of existing relations that may be loaded are displayed. The values of attributes are entered corresponding to each attribute type.

Querying a Database:

A database defined and loaded as specified in the previous paragraphs can be queried. Query will be in the COMMAND mode of interactive session. There are a number of query commands which allow users to query a database. They are described in the following paragraphs.

1. SELECT

Select command is used for displaying data of a relation. Options to display all or selected attributes are available. Several possible select options are given below:

```
SELECT ALL FROM [relation name]
SELECT [attribute name] FROM [relation name]
SELECT ALL FROM [relation name] WHERE [attribute name] [condition]
    [values] [AND/OR] ...
```

continuation dots indicate that upto ten conditions can be specified. The conditions have to the one of the following

```
(a)  [attribute name] [EQ|NE|GT|LT|LE|GE] [value]
(b)  [attribute name] [EQ|NE|GT|LT|LE|GE| [attribute name]
(c)  ROWS [EQ|NE|LT|LE|GE] [row number]
```

2. LISTREL

List command allows user to display information about relations and attributes. The following options are available in LISTREL command.

```
LISTREL
LISTREL [relation name]
LISTREL ALL
```

3. CHANGE

Data values in a relation can be changed using this command. The following options are available

```
CHANGE [attribute] TO [value] IN [relation name]
CHANGE [attribute] TO [VALUE] IN [relation name]
    WHERE ...
```

4. DELETE

This command can be used to delete selected rows in a relation:

```
DELETE ROWS FROM  [relation name] WHERE ...
```

5. RENAME

Attributes and relations can be renamed using this command:

RENAME [attribute] TO [attribute] In [relation name]
RENAME RELATION [relation name] TO [relation name]

6. REMOVE

A relation can be deleted from database definition using REMOVE command:

REMOVE [relation name]

7. PROJECT

This is a relational algebra command. The function of PROJECT is to create a new relation as a subset of an existing relation. The new relation is created from an existing relation by removing attribues, rows or both.

PROJECT [relation name] FROM [relation name]
USING [attribute] [attribute] ... WHERE ...

8. JOIN

The purpose of JOIN command is to combine two relations based on specific attributes from each row. The result of JOIN command is a third relation containing all the specified attributes from both the relations.

JOIN [relation name] USING [attribute name]
WITH [relation name] USING [attribute name]
FORMING [relation name] WHERE ...

9. INPUT

This command assigns input file for MIDAS/R to read data without user interaction

INPUT [file name]

10. OUTPUT

The output of execution may be placed in a given file name. If file name is TERMINAL, then all messages and data are displayed at the terminal:

OUTPUT [file name]

11. EXIT/QUIT

The system buffer data is transfered to database files and database is closed.

12. HELP

User can obtain a description of the available commands on the terminal.

6.2.6 Program Details

MIDAS/R program has about 390 subroutines, 40,000 FORTRAN source statements, and 38 common blocks. Subroutines can be grouped into (i) initialization routines, (ii) file definition routines, (iii) input-output routines, (iv) addressing and searching routines, (v) integrity rule processing routine, (vi) memory management routines, (vii) command processing routine, (viii) relational algebra routines, and (ix) security and protection routines. A brief description of these routines is given in the following paragraphs.

Initialization routines, initialize integer, real and double precision variables. Hollerith constants are assigned to variables. Also, they initialize various common blocks and system buffer arrays. Important initialization routines are RMSTRT, BLKCLN, ZEROIT, RMCONS, and LXCONS.

File definition routines are RMOPEN, RMCLOS, F1OPN, F2OPN, F3OPN, F1CLO, F2CLO, F3CLO, RIOOPN, SETIN, and SETOUT. Each database in MIDAS/R has three files. The first file contains data definition details of relations and attributes. The second file contains actual data. The third file contains keys for locating data. File definition routines open and close database files, assign a database to a logical unit, and assign input and output files.

Input-output routines carryout operations for storing and retrieving data. The main routines for storing and retrieving data are RMPUT and RMGET. They inturn call routines GETDAT and PUTDAT. These routines perform data transfer operations between system buffer and user buffer. At the lowest level, RIOIN and RIOOUT routines actually read and write data in database files. RELGET, RELPUT, RELDEL, and RELADD routines, respectively, get a table from a relation, replace current table from a relation table, delete a current table in a relation table, and add a new tuple to the relation table. Similarly, ATTGET, ATTPUT, ATTDEL and ATTADD routines operate on an attribute table.

Addressing and searching routines determine the physical storage address of data. Relations are stored in random access files. The location of a relation in the data file is specified by a index table. Indicies are assigned to relations by HTOI and IOTH routines. Searching of these indices is done by using binary tree lookup. BTADD, BTGET, BTPUT and BTREP routines add, retrieve, put and replace values in a binary tree. A call to RMFIND and RMWHERE establishes pointers to the required rows of a relation. In turn these routines call RMLOOK, RMSAV, and RMRES routines. These routines also establish pointers to selected attributes of a relation.

Integrity rule processing routines help in maintaining integrity and consistency of a database. Rules specified on attributes of relations are checked by CHKTUP routine. At any stage of execution of DBMS, rule checking flag can be assigned using RMRULE routine. PRULE routine prints out all rules assigned within a database. Rules can be specified using LODRUL routine.

Memory management routines allocate the available computer memory into a number of blocks. The system has capability to allocate 20 blocks or pages at a time. The data is first stored or retreived into these pages. If a page is used completely by a relation then another available page is allocated to the relation. If all pages are occupied then a least recently used page is replaced. The pages not modified are overwritten by a new data. BLKDEF routine defines a new block in memory. BLKEXT allocates size of a block in terms of number of rows and number of columns. BLKCHG changes the dimension of an existing block. BLKMOV moves data between arrays. GLKCLR clears a block from memory.

Command processing routines read interactive commands in a free format, check the syntax of the user commands, interpret them with reference to system conventions and store them for future use. LXLINE routine reads a new command line. LXLENC, LXGETI, LXGETR and LXGETT identify records, integer words, real words and character words in a command line respectively. LXID, LXITEM, LXGEN, and LXEND get identification of the ith item, number of items in the last command, length of a command line, and end of a command line, respectively.

Relational algebra routines perform SELECT, JOIN and PROJECT operations on relations in a database.

Routines for security and protection of a database make password checks, assign passwords and modifies them. Seperate passwords are provided for database modification access and read access. Routines used for this purpose are HASHIN, LODPAS, and RMUSER. HASHIN routine changes 8 character password into a 16 character word. LODPAS processes passwords for relations. RMUSER sets current user identification.

6.2.7 Limitations of MIDAS/R

There are a few limitations of the MIDAS/R program. One of the limitations is that program does not have capability to operate on a number of databases simultaneously. Secondly, the maximum size of a row in a relation cannot exceed 1024 words. If the size of a row exceeds this limit, user should split the row suitably and operate on portions of a row at a time. Also, the number of attributes in a relation cannot exceed 20. The memory management scheme has fixed block size. User has no control over the block size to tune it according to the relation size. At present only five relations can be operated at a time in the system buffer as only five pointers to current relations are maintained by the DBMS. Separate external data modelling facility is not available. User has to operate on the external model which is same as the internal model. This means that there is one-to-one correspondence between external model and internal model.

6.3 IMPLEMENTATION OF MIDAS/N

MIDAS/N is a database management system to support data organization of numerical computations. MIDAS/N is implemented based on numerical data model. In this section, we describe the capabilities, database organization, data definition and data manipulation commands of MIDAS/N. Also details of program architecture and limitations of the system are given.

6.3.1 Capabilities of MIDAS/N

MIDAS/N program is written in FORTRAN 77. It has data definition and data manipulation commands to define large order matrix data and manipulate them. Large matrices such as rectangular, square, upper triangular, lower triangular and hyper matrices can be defined in the database. Matrix data can be arranged in row, column, or submatrix order. Data can be short integer, long integer, real, double-precision and character types. Data mainpulation commands of MIDAS/N can store, retrieve, modify and delete matrices. Data can be accessed in row column or submatrix order. Also individual data elements of a matrix can be accessed. Database security is provided through a password access. Error recovery mechanisms are available.

6.3.2 Database of MIDAS/N

MIDAS/N has capability to create a number of databases, upto a maximum of 20 in the current implementation. These databases can be accessed simultaneously. The databases can be permanent or temporary. A database can store data of a number of matrices, upto a maximum of 20. The size of a database and a matrix is unlimited, but only depends on the availability of disk space. Databases can be organized at a number of hierarchical levels and can be accessed using a path name. Databases and matrices are identified by a unique name. Data organization of MIDAS/N is schematically shown in Fig. 6.3.1.

6.3.3 Data Definition Subroutines of MIDAS/N

Data definition subroutines of MIDAS/N can be used to define databases and matrices. These are FORTRAN call statements and can be directly interfaced with an application program. These are described in the following paragraphs.

Database Definition:

```
CALL NDBDFN (NAME, PTHNAM, TYPE, STAT, IERR)

    NAME   = Name of a database
    PTHNAM = Path name in database hierarchy
    TYPE   = Random or sequential access file type
    STAT   = Permanent or temporary status of a database
    IERR   = Error code
```

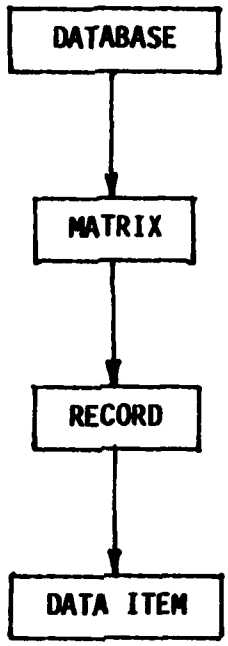


Figure 6.3.1 Logical Data Organization in MIDAS/N

This subroutine can be used to define a database. Path name specifies the hierarchy of databases that are stored in a computer system file directory organized at various levels.

Renaming a Database:

CALL NDBRNM (OLDBN, NEWBN, PTHNAM, IERR)

OLDBN = Old database name
NEWBN = New database name
PTHNAM = Path name
IERR = Error code

This subroutine changes the name of a database.

Matrix Definition:

CALL NDSDFN (NAME, DSNAME, ISUB, JSUB, ORDER, NROW, NCOL, DTYPE, IERR)

NAME = Database name
DSNAME = Data set (Matrix) name
ISUB = Row dimension of a submatrix if present
JSUB = Column dimension of a submatrix if present
ORDER = Order of data storage (explained below)
NROW = Row size of the matrix
NCOL = Column size of the matrix
DTYPE = Data type of data elements in matrix
IERR = Error code

A matrix can be defined by calling this subroutine. Order of the matrix refers to the data storage order which can be row-wise, column-wise, or submatrix-wise. In case of a triangular matrix, order is either row-wise or column-wise. If submatrices are used, then size of a submatrix should be given.

Matrix Redefinition:

CALL NDSRDF (NAME, DSNAME, ISUB, JSUB, ORDER, NROW, NCOL, DTYPE, IERR)

This routine redefines a matrix in a different storage order. A matrix which is in either row, column or submatrix order can be redefined to any of other order (row, column, submatrix). An upper triangular matrix can be redefined to either row or column order. Similarly a lower triangular matrix can be redefined to either row or column order. Data types can be redefined as integer, real and double precision (excepts characters). Arguments are same as in matrix definition.

Matrix Renaming:

CALL NDSRNM (NAME, OLDNAM, NEWNAM, IERR)

NAME = Name of a database
OLDNAM = Old name of a matrix
NEWNAM = New name of a matrix
IERR = Error code

This subroutine changes the name of a matrix.

6.3.4 Data Manipulation Commands

Data manipulation subroutines of MIDAS/N can be used to open, close, delete and compress a database, store, retrieve, delete and copy a matrix. The function and description of these commands are given in the following paragraphs.

Open a Database:

CALL NDBOPN (NAME, PTHNAM, IERR)

NAME = Name of a database
PTHNAM = Path name in database hierarchy
IERR = Error code

This subroutine can be used to open a database.

Close a Database:

CALL NDBEND (NAME, IERR)

NAME = Name of a database
IERR = Error code

This subroutine closes a database. Any modification to data in system buffer are transfered to database files.

Delete a Database:

CALL NDBDEL (NAME, PTHNAM, IERR)

This routine deletes an existing database. Arguments are same as in NDBOPN.

Compress a Database:

CALL NDBCMP (NAME, IERR)

Compresses a database. Empty spaces created due to deletion or redefinition of matrices are removed by moving data in a database. This command helps in efficient utilization of disk space.

Store a Matrix:

CALL NDSPUT (NAME, DSNAME, NSTR, NEND, ISTR, ORDER, UBUF, IROW, ICOL, IERR)

NSTR = Starting row or column or submatrix number for storing data
NEND = Ending row or columing or submatrix number for storing data
ISTR = Starting element number of each row or column
ORDER = Data storage order
UBUF = User buffer (array name)
IROW = Row dimension of the user buffer
ICOL = Column dimension of the user buffer
IERR = Error code

This command stores a matrix data from user buffer into a database. Full or part of a matrix can be stored and its size specified using NSTR and NEND. Row or column storage order can be used for a matrix whose order is defined as row-wise, column-wise or submatrix-wise in data definition. Submatrix storage order can only be used for a matrix defined with submatrix elements.

Retrieve a Matrix:

CALL NDSGET (NAME, DSNAME, NSTR, NEND, ISTR, ORDER, UBUF, IROW, ICOL, IERR)

A matrix can be retrieved into a user buffer from a database using this subroutine. Arguments are the same as defined in NDSPUT. Full or part of a matrix can be retrieved.

Retrieve a Matrix in Row Order:

CALL NDGETR (NAME, DSNAME, NSTR, NEND, ISTR, UBUFF, IROW, ICOL, IERR)

Retrieves a matrix in row order. Arguments are the same as in NDSGET.

Retrieve a Matrix in Column Order:

CALL NDGETC (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Retrieves a matrix in column order. Arguments are the same as in NDSGET.

Retrieve a Matrix in Submatrix Order:

CALL NDGETM (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Retrieves a matrix in submatrix order. Matrix must have been defined to be in submatrix order during data definition. Arguments are the same as in NDSGET.

Store a Matrix in Row Order:

CALL NDPUTR (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Stores a matrix in row order. Arguments are the same as in NDSPUT.

Store a Matrix in Column Order:

CALL NDPUTC (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Stores a matrix in column order. Arguments are the same as in NDSPUT.

Store a Matrix in Submatrix Order:

CALL NDPUTM (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Stores a matrix in submatrix order. Matrix should have been defined to be in submatrix order.

Copy a Matrix:

CALL NDSCPY (NAME1, DSNAME, NAME2, IERR)

NAME1 = Name of the database containing matrix data
DSNAME = Name of the datasset
NAME2 = Name of the database into which matrix has to be copied
IERR = Error code

This subroutine copies a matrix from one database to another database.

Delete a Matrix:

CALL NDSDEL (NAME, DSNAME, IERR)

IERR = Error code

Deletes a matrix from the database.

6.3.5 Matrix Operations Utilities

MIDAS/N has several routines to carry out operations on matrices stored in the database. These include matrix addition, scaling and multiplication routines. Algorithms for these utilities are developed to utilize the storage order of the data sets; i.e., if a matrix is stored in the row order in the database, an algorithm is developed to use the matrix in that order. This is done to minimize the disk I/O and thus perform the operations efficiently. The current routines in the system are described in the following. More routines will be added as need arises.

Multiply General Matrices:

CALL NMTPYx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMTPY1 - Computes $AB = C$
NMTPY2 - Computes $AB^T = C$
NMTPY3 - Computes $A^TB = C$
NMTPY4 - Computes $A^TB^T = C$

Add Matrices:

CALL NMADDx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMADD1 - Computes $A + B = C$
NMADD2 - Computes $A + B^T = C$
NMADD3 - Computes $A^T + B = C$
NMADD4 - Computes $A^T + B^T = C$

Subtract Matrices:

CALL NMSUBx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMSUB1 - Computes $A - B = C$
NMSUB2 - Computes $A - B^T = C$
NMSUB3 - Computes $A^T - B = C$
NMSUB4 - Computes $A^T - B^T = C$

Scale a Matrix:

CALL NMSCLx (NAME1, DSNAME1, NAME2, DSNAME2, SCALE, IERR)

NMSCL1 - Computes A*SCALE = C
NMSCL2 - Computes A^T*SCALE = C

Transpose of a Matrix:

CALL NMTRPZ (NAME1, DSNAME1, NAME2, DSNAME2, IERR)

Computes $A^T = C$

Multiply a Matrix by a Diagonal Matrix:

CALL NMTDGx (NAME1, DSNAME1, NAME2, DSNAME2, ARRAY, IERR)

NMTDG1 - Computes ARRAY*A_T = C
NMTDG2 - Computes ARRAY*A^T
NMTDG3 - Comptes A*ARRAY = C
NMTDG4 - Computes A^T*ARRAY = C

Rearrange Rows/Columns of a Matrix:

CALL NMSRTx (NAME1, DSNAME1, NAME2, DSNAME2, ARRAY, IERR)

NMSRT1 - Rearranges rows according to the order specified in ARRAY
NMSRT2 - Rearranges columns according to the order specified in ARRAY

6.3.6 Equation Solvers and Matrix Decomposition Routines

MIDAS/N has several routines to decompose and solve a linear system of equations. The coefficient matrix may be stored in skyline or banded form. It may also be a genral matrix. In the following, these routines are described. Other equation solvers and eigenvalue extractors will be added at a later date.

Decompose a symmetric matrix by skyline method.

CALL NMSKY1 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MAXCOL, IER)

NAME1 = Database name containing the coefficient matrix

DSNAME1 = Data set name containing the coefficient matrix stored in one dimensional form: the decomposed coefficient matrix is also stored under this name.

NAME2 = Database name containing the addresses of the diagonal elements of the coefficient matrix

DSNAME2 = Data set name containing the addresses of the diagonal elements of the coefficient matrix

NEQ = Size of the coefficient matrix

MAXCOL = Maximum column height of the coefficient matrix

IER = Error parameter

Perform backward and forward substitutions to solve the decomposed system of linear equation by the skyline method.

CALL NMSKY2 (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, NEQ, MAXCOL, IER)

NAME1 = Database name containing the decomposed coefficient matrix

DSNAME1 = Data set name containing the decomposed coefficient matrix stored in one dimensional form

NAME2 = Database name containing the addresses of diagonal elements of coefficient matrix

DSNAME2 = Data set name containing the addresses of diagonal elements of coefficient matrix

NAME3 = Database name containing the R.H.S. vector at entry and solution vector on return

DSNAME3 = Data set name containing the R.H.S. vector at entry and solution vector on return.

NEQ = Size of the coefficient matrix

MAXCOL = Maximum column height of the coefficient matrix

IER = Error parameter

Solve a system of linear equations by the skyline method.

CALL NMSKY3 (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, NEQ, MAXCOL, IER)

NAME1 = Database name containing the coefficient matrix stored in one dimensional form

DSNAME1 = Data set name containing the coefficient matrix stored in one dimensional form at entry, and decomposed coefficient matrix on return

NAME2 = Database name containing the addresses of diagonal elements of coefficient matrix

DSNAME2 = Data set name containing the addresses of diagonal elements of the coefficient matrix

NAME3 = Database name containing R.H.S. vector at entry and solution vector on return

DSNAME3 = Data set name containing R.H.S. vector at entry and solution vector on return

NEQ = Number of equations

MAXCOL = Maximum column height of coefficient matrix

IER = Error parameter

Decompose a symmetric banded matrix by Cholesky's method.

CALL NMBND1 (NAME1, DSNAME1, NEQ, MBND, IER)

NAME1 = Database name containing coefficient matrix at entry and decomposed coefficient matrix on return

DSNAME1 = Data set name containing the coefficient matrix at entry and decomposed coefficient matrix on return

NEQ = Size of the coefficient matrix

MBND = Half bandwidth of the coefficient matrix

IER = Error parameter

Note: The coefficient matrix is banded and is stored in a squeezed form.

Performs backward and forward substitutions to solve decomposed system of linear banded equations.

CALL NMBND2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MBND, IER)

NAME1 = Database name containing decomposed coefficient matrix

DSNAME1 = Data set name containing decomposed coefficient matrix

NAME2 = Database name containing R.H.S. vector at entry and solution vector on return

DSNAME2 = Data set name containing R.H.S. vector at entry and solution vector on return

NEQ = Size of coefficient matrix

IER = The decomposed matrix is stored in a squeezed form

Note: The decomposed matrix is stored in a squeezed form.

Solve system of linear banded equation by Cholesky's method.

CALL NMBAND3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MBND, IER)

NAME1 = Database name containing the coefficient matrix at entry and decomposed matrix on return

DSNAME1 = Data set name containing the coefficient matrix at entry and decomposed matrix on return

NAME2 = Database name containing the R.H.S. vector at entry and solution vector on return.

DSNAME2 = Data set name containing the R.H.S. vector at entry and solution vector on return

NEQ = size of the coefficient matrix

MBND = Half bandwidth of the coefficient matrix

IER = Error parameter

Decompose a general full matrix.

CALL NMGSL1 (NAME1, DSNAME1, NEQ, IER)

NAME1 = Database name containing the coefficient matrix at entry and decomposed matrix on return

DSNAME1 = Data set name containing the coefficient matrix at entry and decomposed matrix on return

NEQ = Size of the coefficient matrix

IER = Error parameter

Perform backward and forward substitutions to solve a decomposed general system of equations.

CALL NMGSL2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

NAME = Database name containing the decomposed coefficient matrix

DSNAME1 = Data set name containing the decomposed coefficient matrix

NAME2 = Database name containing the R.H.S. vector at entry and solution vector on return

DSNAME2 = Data set name containing the R.H.S. vector at entry and solution vector on return

NEQ = Size of coefficient matrix
IER = Error parameter

Solve system of linear equations.

CALL NMGSL3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

NAME1 = Database name containing the coefficient matrix at entry and decomposed matrix on return

DSNAME1 = Data set name containing the coefficient matrix at entry and decomposed matrix on return

NAME2 = Database name containing the R.H.S. vector at entry and solution vector on return

DSNAME2 = Data set name containing the R.H.S. vector at entry and solution vector on return

NEQ = Size of coefficient matrix

IER = Error parameter

Decompose a full symmetric matrix by modified Choleshy's method.

CALL NMSYM1 (NAME1, DSNAME1, NEQ, IER)

NAME1 = Database name containing the coefficient matrix at entry and decomposed matrix on return

DSNAME1 = Data set name continaing the coefficient matrix at entry and decomposed matrix on return

NEQ = Size of coefficient matrix

IER = Error parameter

Perform backward and forward substitution to solve a decomposed symmetric system of linear equations.

CALL NMSYM2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

NAME1 = Database name containing the decomposed coeffcient matrix

DSNAME1 = Data set name containing the decomposed coefficient matrix

NAME2 = Database name containing R.H.S. vector at entry and solution vector on return

DSNAME2 = Data set name containing R.H.S. vector at entry and solution vector on return

NEQ = Size of the coefficient matrix

IER = Error parameter

Solve a full symmetric system of linear equations by the modified Cholsky's method.

CALL NMSYM3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

NAME1 = Database name containing the coefficient matrix, at entry and decomposed matrix on return

DSNAME1 = Data set name containing the coefficient matrix at entry and decomposed matrix on return

NAME2 = Database name containing R.H.S. vector, at entry and solution vector on return

DSNAME2 = Data set name containing R.H.S. vector at entry and solution vector on return

NEQ = Size of coefficient matrix

IER = Error parameter

6.3.7 Program Details

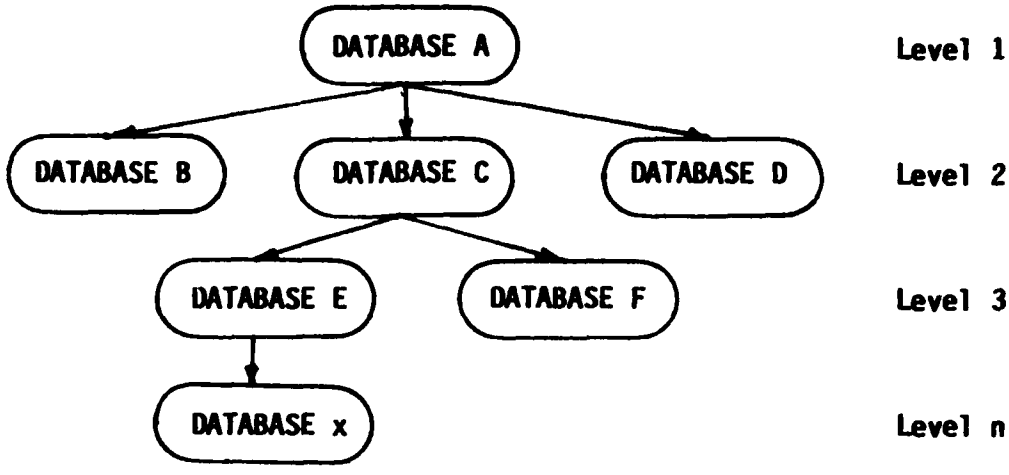
MIDAS/N is written in FORTRAN 77. Subroutines of the program can be grouped into (i) file definition routines, (ii) input-output routines, (iii) addressing routines, and (iv) memory management routines. A brief description of these routines is given in the following paragraphs.

File definition routines open a file unit and assign database name to be the file name. An available logical unit number is assigned to the file. Routine NDBDFN performs this function. File status and file types are assigned according to user request. File organization is shown in Fig. 6.3.2.

Input-output routines perform data storage and retrieval operation. NDSPUT, NDSGET, IN$MEM, IN$DST, D$ASIN, D$READ and D$WRITE routines do input-output operations. These routine transfer data from user buffer to system buffer and vice-versa. Also these routines check the matrix order, data type and matrix size and prints out error message if data manipulation operations are not valid.

Addressing routines IN$MEM and IN$DST allocate physical storage location address to matrices created in the database. The system maintains an index table to provide address of stored records. Index table provides a pointer to data defintion block which contains details of a matrix such as name, type, order, size. Data definition block is also stored at the beginning of actual data in a file. Matrix data is mapped on to physical storage space in a linear address sequence. Smallest physical data item correspond to one word length. The physical storage structure is schematically shown in Fig. 6.3.3.

Memory management routines G$PAGE, R$MVPG and P$EXST allocate pages in the memory to various matrices. The scheme uses fixed number of pages of same size. The paging memory is an array in the common block MCONTNT of short integer variable. FORTRAN equivalence statement is provided to deal with other data types. The memory management scheme uses 'Least Recently Used' page replacement algorithm. A counter is maintained for each page. When a page is to be replaced the page having highest counter value becomes the



Path name to database F ≡ A > C > F

Figure 6.3.2 Hierarchical Level of Database Organization

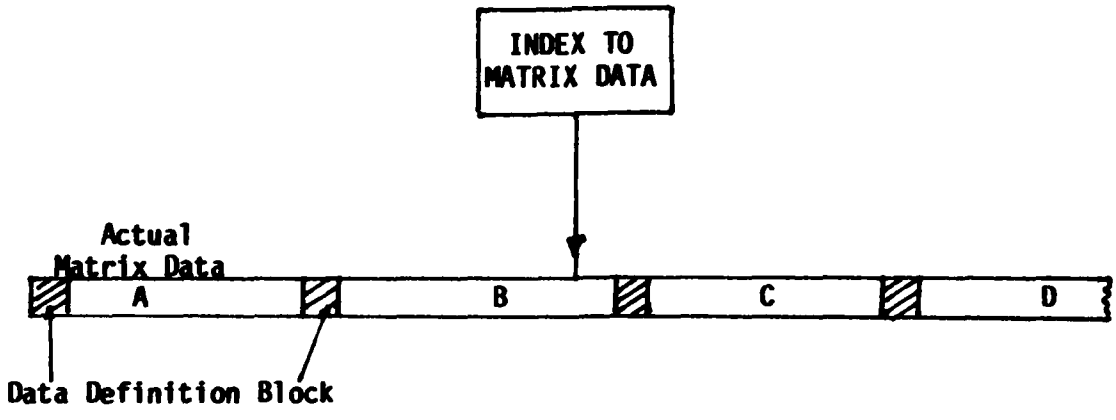


Figure 6.3.3 Physical Storage Structure

candidate. Page replacement is done when no free pages are available. Page not modified is overwritten instead of replacement.

6.3.8 Limitations of the MIDAS/N

There are a few limitations in the system. Matrix size remains fixed after it is defined. Any alteration in matrix size requires data transfer to a new physical storage location. At present maximum of 20 matrices can be defined in a database. However this number can be increased by changing certain parametes in the program. The system does not support external data modelling facility. Thus, application program view of database is tied to the internal data model.

7. SUMMARY, DISCUSSION AND CONCLUSIONS

7.1 Summary

Computer-aided structural design means integration of structural engineering design methods and computer-science in a computer-based system containing a database, a program library and a man-machine communication link. With this definition in view, a new concept was presented for integrating finite element analysis and design optimization methodology into a computer-based system. Emphasis was placed upon database management concept for structural design. Several reasons exist for emphasizing data management in design. First, the iterative nature of optimal design process uses large amount of data for computation. Secondly, existing finite element programs are not flexible to use modified data generated at various stages of design. Thirdly, designer needs control over the program and data to obtain optimum design. Finally, a good database will enable addition of new optimization and other programs without extensive modification of database or existing programs. Also, several designers can be allowed to use a common database to investigate alternate designs.

Structural design process was described to bring out various steps involved in design of structures. Mathematical modeling of the design process was presented to describe the nature of computation and data used in design. Important components required to build a computer-aided structural design system were described. Need for a good data management system was emphasized. Users of computer-aided structural design system was identified and their requirements were described.

A study of database management concepts applicable to finite element analysis and structural design optimization was conducted. This study was essential since the data managements concepts are relatively new to engineering community. Definition of various terminologies was given and described with reference to examples from finite element analysis and optimization data. Hierarchical, network and relational data models were described. The advantages and disadvantages of these models were given. Relational data model was found to be more appropriate for structural data organization. The concept of normalization of data was described. This concept provides certain guidelines to group different data items to form associations. Importance of maintaining integrity of databases was emphasized. Global and local database concepts were described and their use of design optimization was brought out.

A methodology to design a structural design database was proposed. Till now, no such methodology was used for data organization of existing finite element programs, since no scientific database management techniques were available at the time those programs were developed. Three levels of data organization -- conceptual, internal and external are suggested for structural design databases. Data organization at the conceptual level represents inherent characteristics of data regardless of whether or not database management software available supports such organization directly. Various steps were identified to develop a conceptual data model. Methodology for including vector and matrix data in the conceptual data model was described. A methodology for constructing an internal model was proposed. The internal model aims to store the structural design data in an efficient way.

Consideration for reducing storage space and data access time was made. Normalization of data was suggested to avoid various anomalies in storage operation. Method for developing an external model was given. An external data model enables us to provide data to several application programs depending on their needs. One of the important aspects in the design of database for structural analysis and optimization is the need to accomodate large matrix data. A methodology was developed to store large matrix data in the database. Various types of large matrices -- square, triangular, banded, hypermatrix, skyline matrix -- were identified and their characteristics were studied. Various aspects like storage space, processing sequence, matrix operation, page size, flexibility of modification, etc., were considered to develop suitable storage schemes. A numerical model was developed to represent large order matrix data. Finally, an algorithmic model was proposed to deal with storage and computation efficiency aspect required for structural design optimization programs.

A proposal to develop a database management system for structural analysis and optimal design was made. Components required for a good database management system were described. Some important components of the database management system are -- languages, command processors, addressing and searching, file definition and file operation, integrity rule processor, memory management and security and protection routines. Considerations for developing data definition and data manipulation languages were given. Query language was proposed for an interactive user of a database. A syntax (grammer) for these languages was given.

Implementation of a database management system -- MIDAS was done. MIDAS program has two subsystems -- MIDAS/R, MIDAS/N. MIDAS/R is based on relational model of data, MIDAS/N is based on numerical model of data. MIDAS/R relieves the burden of managing data for application programmers by providing user-friendly application commands. The program has sophisitcated interactive commands to query the database. Relations can be defined and manipulated using data definition and data manipulation commands of MIDAS/R. Interactive commands of MIDAS/R can also be used to define and manipulate relations in a database. Relational algebra commands -- SELECT, PROJECT and Join are available to manipulate the relations. MIDAS/R has capability to store any number of relations in a database. Numerical database management system -- MIDAS/N has capability to store matrix data of finite element analysis and optimization programs. MIDAS/N has capability to create a number of databases upto a maximum of 20. These databases can be accessed simultaneously. Matrices in the database can be stored and accessed in row, column and submatrix order. Various types of matrices -- square, rectangular, triangular and hypermatrix can be organized in the database of MIDAS/N. In addition to database management functions, MIDAS/N has several routines to solve equations and decompose matrices.

7.2 Discussion

The study answers several problems facing data management in finite element analysis and optimization problems. The following questions were addressed: (i) How the database has to be organized? (ii) What kind of information is to be stored? (iii) What kind of database management system is suitable? (iv) How data is manipulated? (v) How various applications use

the data? Answers to these questions were not available in the structural design field as only a few research studies have been done on the topic. Thus, there was a need to introduce new concepts many of which are being researched in computer-science and business data management fields. However, the concepts were developed only for business data management application and they were not directly applicable to engineering data management needs. Therefore, an attempt was made to study all the relevant data management concepts. The concepts that were found applicable were described with reference to examples from finite element analysis and design optimization data.

Even though many sophisticated finite element programs are available, they use only a primitive data organization routines. Several reasons exist for non-availability of suitable data management systems in these programs. First, database management techniques were not well understood at the time these programs were developed. Secondly, finite element programs were developed to be treated like a block box with certain input and output. No provision was made in these programs to provide access to intermediate data generated by the programs. Finally iterative analysis of structures was not considered as design optimation techniques have been developed only recently.

Thus, database management approach presented in the study offers solution to many of these problems. The study shows how data of finit element analysis and optimization can be organized using various data models. Out of the three data models -- hierarchical, network and relational -- the latter was found to be more appropriate. The reasons being, relational model is simple, easy to use and all the characteristics of structural design data can be represented in the model. Therefore, relational model was selected for a more detailed study. Questions were posed as to how to decide what data items have to be grouped together? In particular using a relational model, how do we determine what relations are needed and what their attributes should be? It was shown that normalization of data provides certain guidelines to group data items together to form relations. First, second and third normal forms of data were illustrated. The concepts of semantic integrity and consistency, transaction management, and global and local database netowrks were described with reference to structural design applications. However, some of the concepts in transaction management are new and of theoretical research interest. Program (technical) realization of integrity concepts are yet to be seen.

Till now data organization in finite element programs was based on intution, since no methodology was available to design a database. The study proposes a methodology to design a database. Three levels of data organization -- conceptual, internal and external levels -- were proposed. This method provides clear distinction between theoretical and implementation aspect of data organization. Conceptual data model is of theoritical nature and represent the inherent characteristics of data. Internal model can be independently developed to provide the storage and access time efficiency. Later, correctness of internal model that is to be implemented on a computer system can be verified using the conceptual model. In the methodology proposed, there is flexibility to choose an external model which is same as an internal model, or develop different external models according to needs of applications. More study is required to give a clear idea of how an external

model can be provided to answer the update problems. Numerical model provides a scheme to organize large matrix data. Various suggestions made to organize matrices are suitable for implementation. Efficiency of storage space and accessing time can be realized using this model. Algorithmic model further enables us to conserve storage space for applications like generation of element stiffness matrices instead of storing them in a database. Thus, the methodology enables us to design a good database for structural design applications.

In the second part of the study, we dealt with the need of a software for database management. What kind of database management program is suitable? It is possible to use existing DBMS or develop a new database management system? What modifications are required to existing DBMS so that it can be used in structural design application? These questions were tackled by a detailed study of requirements of a database management. It was realized that data definition language, data manipulation language play an important role in providing a communication link between designer and computer system. Syntax (grammer) for these languages was suggested to enable suitable database management program development. Memory management schemes suggested in the study are useful for efficient utilization of large memory in a computer system. Review of several database management programs has shown that most of the programs are not directly applicable to organize sructural design data.

MIDAS program developed was found to be very useful for data management. The program relieves the burdon of managing data for application programmers. Both relations and matrix data can be organized. The command in the program are simple to use and provide sophisticated capability to the user. They are capable of storing, deleting and modifying data in the database. Interactive capability of the program is useful for the designer to change design parameters. The program satisfies requirements of the specified database management system. The program is well documented and can be easily be easily incorporated into structural analysis and design optimization programs. This work is in progress and will be reported at a later date.

7.3 Conclusions

A new approach was presented in the report to integrate structural design methods and computer-science concepts to provide a computer-based system for analysis and optimization of structural systems. Several problems of data organization for finite element analysis and optimization application can be overcome by providing sophisticated database management systems. The study of various database management concepts has shown that they are applicable to data organization in structural design area. We can deal with special characteristics of structural design data by suitably modifying and extending these concepts. The methodology presented for designing a database for structural applications is useful and this methodology will replace the intutive way of data organization for finite element analysis and optimization programs. The study has identified the requirements of a database management programs. Syntax required for data definition, data manipulation and query languages was developed and are useful for providing a good communication link between designer and computer. A sophisticated database management system -- MIDAS was implemented. The program can be used either through an application program or interactively. MIDAS is very useful to manage data of structural

analysis and design optimization applications. It is concluded that with the proposed database design methodology and the advanced database management system, optimal design of complex systems of today can be attempted. The basic tools described and developed in the study will facilitate in making this a reality.

Appendix 1

An algorithm for determining transitive closure from a given connectivity matrix C is as follows:

1. Form matrix J with
   $J(i,i) = 0$ for $1 \leq i \leq n$
   $= 1$ for $i \neq j$

2. Form modified connectivity matrix CC
   with $CC(i,j) = C(i,j)$ $1 \leq i \leq n$
   $1 \leq j \leq n$
   and $CC(i,i) = 0$

3. DO i = 1,n
   DO k = 1,n
   DO j = 1,n
   If If (CC(i,j)•EQ•1•AND•CC(j,k)•EQ•1)C(i,k) =1
   End DO loop

4. Remove erroneous dependencies that were derived from the situation $N_i \rightarrow N_j \rightarrow N_i$

5. IS new matrix C obtained? If no, stop

6. Form modified connectivity matrix (as in step 2) using CC matrix derived in step 3

7. Go to step 3

8. End

APPENDIX 2

BNF Description of the Proposed Data Definition Language

```
<letter>:: = A|B|C|•••••|Z
<digits>:: = 1|2|•••••9|0
<basic symbols:: = <letters>|<digits>
<string>:: = <any sequence of basic symbols>
<variable>:: = <simple variable>|<subscripted variable>
<simple variable>:: = <identifier>
<identifier>:: = <letter>|<identifer><letter or digit>
<subscripted variable>:: = <identifer>[<subscript list>]
<subscript list>:: = <fortran subscript list>
<empty>:: = <null string of symbols>
<unsigned integer>:: = <digit>|<unsigned integer><digit>

<data definition statement>:: =
                        <database definition statement>|
                        <database user specification statement>|
                        <database definition statement>|
                        <relation data definition statement>|
                        <numerical data definition statement>|
                        <data definition termination statement>|
                        <data redefinition statement>

<database definition statement>:: =
                        <reserved procedure DBDEFN>
                        (<database definition parameter part>)
<database definition parameter part>:: =
                        <database name>
                        <database hierarchy>
                        <database status>
                        <database definition error code>
<database name>:: = '<name>'|<variable>
<database heirarchy>:: = '<heirarchy type>'|<variable>
<hierarchy type>:: = GLOBAL|LOCAL
<name>:: = <letter><string>
<database status>:: = '<status type>'|<variable>
<status type>:: = PERMANENT|TEMPORARY
<database definition error code>:: = <variable>
```

Note: 1) Vertical bar | denotes options for choosing items to the left of the bar or to the right of the bar
2) [ ] indicate items within it are optional
3) <x>:: = <y> | <x><z> denotes a recursive statement. x is used repeatedly
4) ' ' indicates items within it taken as data

```
<database user identification statement>:: =
                        reserved procedure DBUSER>
                        (<database user identification parameter part>)
<database user identification parameter part>:: =
                        <database name>,
                        <database user type>,
                        <database access type>, <password>,
                        <user identification error code>
<database user type>:: = '<user type>'|<variable>
<user type>:  = OWNER|USER
<database access type>:: = '<access type>'|<variable>
<access type>:: = READ|MODIFY
<password>:: = <letter><string>
<user identification error code>:: = <variable>

<data set definition statement>:: =
                        <reserved procedure DSDEFN>
                        (<dataset definition parameter part>)
<data set definition parameter part>:: =
                        <database name>,
                        <dataset name>,
                        <dataset type>,
                        <data item row size>,
                        <data item column size>,
                        <data set definition error code>
<data set name>:: = '<name>'|<variable>
<data set type>:: = '<type specification>'|<variable>
<type specifiation>:: = INT|REAL|DOUB
                        IVEC|RVEC|DVEC
                        IMAT|RMAT|DMAT
<data item row size>:: = <size>|<empty>|VAR
<data item column size:: = <size>|<empty>|VAR
<size>:: = <unsigned integer>|<variable>
<data set definition error code>:: = <variable>

<relation definition statement>:: =
                        <reserved procedure DRLDFN>
                        (<relation definition parameter part>)
<relation definition parameter part>:: =
                        <database name>,
                        <relation name>,
                        <attribute name array>,
                        <attribute type array>,
                        <attribute row size array>,
                        <attribute column size array>,
                        <attribute key specification array>,
                        <relation definition error code>
<relation name>:: = '<name>'|<variable>
<attribute name array>:: = '<name array>'|<variable>
<name array>:: = <name>|<name><name array>
<attribute type array>:: = '<type specification array>'|<variable>
<type specification array>:: = <type specification>
                        |<type specification><type specification array>
<attribute row size array>:: = <variable>
```

```
<attribute column size array>:: = <variable>
<attribute key specification array>:: =
                        <key specification array>|<variable>
<key specification array>:: = KEY|KEY<key specification>|<empty>
<relation definition error code>:: = <variable>

<data definition termination statement>:: =
                        <reserved procedure DSEND>
                        (<data definition termination parameter part>)
<database definition parameter part>:: = <empty>

<data redefinition statement>:: =
                        <reserved procedure DSRDFN>
                        (<data redefinition parameter part>)
<data redefinition parameter part>:: =
                        <data set definition parameter part>
                        |<relation definition parameter part>
                        |<numerical data definition parameter part>

<matrix data definition statement>:: =
                        <reserved procedure DSMATX>
                        (<matrix data definition parameters part>)

<matrix data definition part>:: =
                        <database name>
                        <matrix identification>
                        <matrix characteristics>
                        <matrix definition error code>
<matrix identification>:: = '<name>'|<variable>
<matrix characteristics>:: =
                        'SQUARE', <S. Details>|
                        'BANDED', <B. Details>|
                        'HYPERMATRIX', <H. Details>|
                        'SKYLINE, <K. Details>|
                        'SPARSE', <P. Details>

<S. Details>:: =        <matrix storage type>
                        <matrix ordering>,
                        <matrix size>,
                        <empty>, <empty>,
                        <empty>, <empty>,
                        <empty>, <empty>,

<matrix storage type>:: = '<matrix storage string>'|<variable>
<matrix storage string>:: = UPPER|LOWER|FULL
<matrix ordering>:: = '<matrix ordering string>'|<variable>
<matrix ordering string>:: = ROW|COLUMN
<matrix size>:: =       <row size>,<column size>|VAR,<column size>|
                        row size>,VAR|VAR,VAR

B. Details>:: =         <matrix storage type>,
                        <matrix ordering>,
                        <matrix size>,
                        <matrix band size>,
```

```
<matrix band size>:: = <number of upper codiagonals>,
                       <number of lower codiagonals>
<number of upper codiagonals>:: = <unsigned integer>
<number of lower codiagonals>:: = <unsigned integer>

<H. Details>:: =       <matrix storage type>
                       <matrix ordering>,
                       <matrix size>,
                       <submatrix size>
<submatrix size>:: = row size>|<variable>,<column size>|<variable>

<K. Details>:: =       <empty>,
                       <empty>,
                       <matrix size>,
                       <skyline definition>

<skyline definition>:: = <array of skyline height>

<P. Details>:: =       <empty>,
                       <empty>,
                       <matrix size>,
                       <empty>,<empty>
                       <empty>,<empty>
```

APPENDIX 3

BNF Description of the Proposed Data Manipulation Language

```
<Data manipulation statement>:: =
                        <database open statement>
                        <database close statement>
                        <data retrieval statement>
                        <data append statement>
                        <data modify statement>
                        <data delete statement>
                        <data copy statement>
                        <matrix retrieval statement>
                        <matrix append statement>
                        <matrix modify statement>
                        <matrix delete statement>
                        <matrix copy statement>

<database open statement>:: =
                        <reserved procedure DBOPEN>
                        (<database open parameter part>
<database open parameter part>:: =
                        <database name>,
                        <database hierarchy>
                        <database open error code>
<database open error code>:: = <variable>

<database close statement>:: =
                        <reserved procedure DBCLOS>
                        (<database close statement>)
<database close statement>:: =
                        <database name>
                        <database hierarchy>
                        <database close error code>

<data retrieval statement>:: =
                        <reserved procedure DSGET>
                        (<data retrieval parameter part>)

<data retrieval prarmeter part>:: =
                        <database name>,
                        <data set name>|<relation name>,
                        <identification number>|<empty>,
                        <user buffer>,
                        <data manipulation error code>

<identification number>:: = <tuple number>|<row number>
<tuple number>:: = <unsigned integer>|<variable>
<row number>:: = <unsigned integer>|<variable>
<user buffer>:: = <variable>
<data manipulation error code>:: = <variable>
<data append statement>:: =
```

```
                        <reserved procedure DSPUT>
                        (<data append parameter part>)

<data append parameter part>:: =
                        <data retrieval parameter part>

<data modify statement>:: =
                        <reserved procedure DSMOD>
                        (<data modify parameter part>)

<data modify parameter part>:: =
                        <data retrieval parameter part>

<data delete statement>:: =
                        <database name>,
                        <data set name>|<relation name>,
                        <identification number>|<empty>,
                        <data manipulation error code>

<data copy statement>:: =
                        <reserved procedure DSCOPY>
                        (<data copy parameter part>)

<data copy parameter part>:: =
                        <database name-copy from>,
                        <data set or relation name-copy from>
                        <database name-copy to>
                        <data set or relation name-copy to>
                        <data manipulation error code>

<database name-copy from>:: = <database name>
<data set or relation name-copy from>:: = <data set name>|<relation name>

<database name-copy to >:: = <database name>
<data set or relation name - copy to>:: = <data set name>|<relation name>
<data manipulation error code>:: = <variable>

<matrix retrieval statement>:: =
                        <reserved procedure MTGET>
                        (<matrix retrieval parameters part>)

<matrix retrieval parameter part>:: = <database name>,
                        <matrix identification>,
                        <row number>,[<column number>],
                        |<column number>,[<row number>],
                        |<empty>,<empty>,
                        <user buffer>,
                        <matrix manipulation error code>
<matrix manipulation error code>:: = <variable>

<matrix append statement>:: =
                        <reserved procedure MTPUT>
                        (<matrix append parameter part>)
<matrix append parameter part>:: =
```

```
                        <matrix retrieval parameter part>

<matrix modify statement>:: =
                        <reserved procedure MTMOD>
                        (<matrix modify parameter part>)
<matrix modify parameter part>:: =
                        <matrix retrieval parameter part>
<matrix delete statement>:: =
                        <reserved procedure MTDEL>
                        (<matrix delete parameter part>)

<matrix delete parameter part>:: =
                        <database name>
                        <matrix identification>
                        <row number>,[<column number>],
                        |<column number>,[<row number>],
                        |<empty>,<empty>,
                        <matrix manipulation error code>

<matrix copy statement>:: =
                        <reserved procedure MTCOPY>
                        (<matrix copy parameter part>)
<matrix copy parameter part>:: =
                        <database name-copy from>,
                        <matrix identification-copy from>,
                        <database name-copy to>,
                        <matrix identification-copy to>,
                        <matrix manipulation error code>
<matrix identification-copy from>:: = <matrix identification>
<matrix identification-copy to>:: = <matrix identification>
```

REFERENCES

Allan III, J.J., 1972, "Foundations of the Many Manifestations of Computer Augmented Design," Computer-Aided Design, Proceedings of International Federation of Information Processing, pp. 27-58.

Arora, J.S. and Govil, A.K., 1977, "An Efficient Method for Optimal Structural Design by Substructuring," Computers and Structures, Vol. 7, pp. 507-515.

Arora, J.S., Ryu, Y.S. and Wu, C.C., 1984a, "A User's Manual for the Computer Program DOCS: Level 3.0," Applied-Optimal Design Laboratory, The University of Iowa.

Arora, J.S., Thanedar, P.B., Tseng, C.H. and Hwang, R.S., 1984b, "User's Manual for Program IDESIGN Version 3.1," Applied-Optimal Design Laboratory, The University of Iowa.

Afimiwala, K.A. and Mayne, R.W., 1979, "Interactive Computer Methods for Design Optimization," Computer-Aided design, Vol. 11, No. 4, pp. 201-208.

Bell, Jean, 1982, "Data Modelling of Scientific Simulation Programs," Int. Conf. On Management of data, ACM-SIGMOD, pp. 79-86.

Bennett, J.A. and Nelson, M.F., 1979, "An Optimization Capability for Automotive Structures," SAE Transactions, Vol. 88, pp. 3236-3243.

Blackburn, C.L., Storaasli, O.O. and Fulton, R.E., 1982, "The Role and Application of Database Management in Integrate Computer Design," Journal of American Institute of Aeronautics and Astronautics, pp. 603-613.

Browne, J.C., 1976, "Data Definition, Structures, and Management in Scientific Computing," Proc. Of ICASE Conference on Scientific Computing, pp. 25-56.

Bryant, J.C., 1978, "A Data Management System for Weight Control and Design-to-Cost", NASA Conference Publication 2055. pp. 65-84.

Buchmann, A.P. and Dale, A.G., 1979, "Evaluation Criteria for Logical Database Design Methodologies," Computer-Aided Design, pp. 121-126.

Comfort, D.L. and Erickson, W.J., 1978, "RIM-A Prototype For A Relational Information Managemer System," NASA Conference Publications 2055, pp. 183-196.

Czekalinski, L. and Zgorzelski, M., 1982, "Design Database Organization and Access Problems in Large Scale Machine Manufacturing Industry," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 297-308.

Daini, O.A., 1982, "Numerical Database Management System: A Model," Int. Confer. On Management Of Data ACM-SIGMOD.

Darby-Dowman, K. and Mitra, G., 1983, "Matrix Storage Schemes In Linear Programming," SIGMAP bulletin ACM, No.32, pp. 24-38.

Date, C.J., 1977, An Introduction To Database Systems, Addison-Wesley, Reading, Mass., 1977.

Derwa, G.T., 1978, "Advanced Program Weight Control System," NASA Conference Publication 2055, pp. 55-64.

Eastman, C.M. and Henrion, M., 1980, "The Glide Language for CAD," J. Of the Technical Councils Of ASCE, Vol. 106, No. TC1, pp. 171-184.

Eastman, C.M. and Fenves, S.J., 1978, "Design Representation and Consistency Maintenance Needs in Engineering Databases," NASA Conference Publication 2055, pp. 1-18.

Eastman, C.M., 1978, "The Representation of Design Problems and Maintenance of Their Structure," Artificial Intelligence and Pattern Recognition in Computer-Aided Design, Proceedings of International Federation of Information Processing, pp. 335-366.

Eberlein, W. and Wedekind, H., 1982, "A Methodology for Embedding Design Databases into Integrated Engineering Systems," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 3-37.

Elliott, L., Kunii, H.S., and Browne J.C., 1978, "A Data Management System For Engineering and Scientific Computing," NASA Conference Publications 2055, pp. 197-222.

Emkin, L.Z., 1978, "ICES Cocepts-A Modern System Approach," Computing In Civil Engineering pp. 89-107.

Encarnacão, J. and Schlechtendahl, E.G., 1983, Computer-Aided Design, Springer-Verlag, Berlin.

Felippa, C.A., 1979, "Database Management In Scientific Computing-I General Description," Computers and Structures, Vol. 10, pp. 53-61.

Felippa, C.A., 1980, "Database Management In Scientific Computing-II, Data structures and Program architecture," Computers and Structures, Vol. 12, pp. 131-145.

Felippa, C.A., 1982, "Fortran-77 Simulation of Word-addressable Files," Advances in Engineering software, Vol. 4, No. 4, pp. 156-162.

Fischer, W.E., 1979 "PHIDAS -a Database Management System for CAD/CAM Software," Computer-Aided Design, Vol 11, No. 3, pp. 146-150.

Fishwick, P.A. and Blackburn, C.L., 1982, "The Integration Engineering Programs using a Relational Database Scheme," Computers In Engg, Int. Comp. Engg. Confer., pp. 173-181.

Fleury, C., Ramanathan, R.K., Salana, M. and Schmit, Jr., L.A., 1981, "ACCESS Computer program for the Synthesis of Large Structural Systems," Proceedings of the International Symposium on Optimum Structural Design, University of Arizona, Tucson.

Foisseu, J. and Valette, F.R., 1982, "A Computer Aided Design Data Model: FLOREAL," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 315-330.

Fulton, R. E. and Voigt, S.J., 1976, "Computer-Aided Design and Computer Science Technology," Third ICASE conf. on Scientific Computing, pp. 57-82.

Galletti, C.U. and Giannotti, E.I., 1981, "Interactive Computer System Functional Design Of Mechanisms," Computer-Aided Design, Vol. 13, No. 3, pp. 159-163.

Giles, G.L. and Haftka, R.T., 1978, "SPAR Data Handling Utilities," NASA Technical Memorandum 78701.

Grabowski, H. and Eigner, M., 1982, "A Data Model for a Design Database," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 117-144.

Grabowski, H., Eigner, M. and Rausch, W., 1978, "CAD Data-Structures For Minicomputers," Third Int. Conf On Computers and Engg., pp. 530-548.

Grabowski, H. and Eigner, M., 1979, "Semantic Datamodel Requirements and Realization with a Relational Data Structure," Computer-Aided Design, Vol. 11, No. 3, pp. 158-167.

Haskin, R.L. and Lorie, R.A., 1982, "On extending the Functions Of a Relational Database System," Int. Conf. On Management of Data ACM, pp. 207-212.

Haug, E.J. and Arora, J.S., 1979, "Applied Optimal Design," John Wiley and Co..

Heerema, F.J. and van Hedel, H., 1983, "An Engineering Data managemet System for Computer-Aided Design", Advanced in Engineering Software, Vol. 5, No. 2, pp. 67-75.

Jefferson, D.K. and Thomson, B.M., 1978, "Engineering Data Management: Experience and Projections," NASA Conference publication 2055, pp. 223-242.

Jenne, R.L. and Joseph, D.H., 1978, "Management of Atmospheric Data", NASA Conference Publication 2055, pp. 129-140.

Johnson, H.R., Comfort, D.L. and Shull, D.D., 1980, "An Engineering Data Management System for IPAD," IPAD: Integrated Programs for Aerospace-vehicle Design, NASA Conference Publication 2143, pp. 145-178.

Jumarie, G., 1982, "A Decentralized Database via Micro-computers a Preliminary Study," Computers in Engineering, Int. comp. Engg. Confer. ASME, pp. 183-187.

Kamel, H.A., McCabe, M.W. and Spector, W.W., 1979, GIFTS5 System Manual", University of Arizona, Tucson.

Koriba, M., 1983, "Database Systems: Their Applications to CAD Software Design," Computer-Aided Design, Vol. 15, No. 5, pp. 277-288.

Kunni, T.I. and Kunni, H.S., 1979, "Architecture of a Virtual Graphic Database System For Interacive CAD", Computer-Aided Design, Vol. 11, No. 3, pp. 132-135.

Kutay, A.R. and Eastman, C.M., 1983, "Transaction Management in Engineering Databases," Engineering Design Aplications, Proceedings of Annual Meeting, Database Week, ACM SIGMOD, pp. 73-80.

Lafue, G., 1978, "Design Database and Data Base Design," Third Int. Conf. On compters in Engg. and Building Design CAD78, Brighton Metropole, Sussex, U.K., 14-16.

Lafue, G.M.E., 1979, "Integrating Language Database for CAD Applications," Computer-Aided Design, Vol. 11, No. 3, pp. 127-129.

Leinemann, K. and Schlechtendahl, E.G., 1976, "The Regent System for CAD," CAD Systems, Proceedings of International Federation of Information Processing, pp. 143-168.

Lillehagen, F.M., and Dokken, T., 1982, "Towards a Methodology for Constructing Product Modelling Databases in CAD," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 59-88.

Lopatka, R.S. and Johnson, T.G., 1978, "CAD/CAM Data Management Needs, Requirements and Options," NASA Conference Publications 2055, pp. 25-40.

Lopez, L.A., 1974, "FILES: Automated Engineering Data Management System," Computers in Civil Engineering, Electronic Computation, pp. 47-71.

Lopez, L.A., Dodds, R.H., Rehak, D.R. and Urzua, J.L., 1978, "Application of Data Management to Structures," Computing in Civil Engineering, pp. 477-498.

Managaki, M., 1982, "Multi-layered Database Architecture for CAD CAM Systems," File Structures and Databases for CAD, Proceedings of International Federation of Information Processing, pp. 281-290.

Martin, J., 1977, "Computer Database Organization", Prentice-Hall. Inc., Englewood Cliff, N.J.

Massena, W.A., 1978, "SDMS - A Scientific Data Management System," NASA Conference Publication 2055, pp. 143-154.

Nye, W., 1981, "DELIGHT- Design Language with Interactive Graphics and a Happier Tomorrow," Electronics Research Laboratory, University of California, Berkeley, CA.

Pahl, P.J., 1981, "Data Management in Finite Element Analysis," Nonlinear Finite Element Analysis in Structural Mechanics, Wunderlich, W., Stein, E. and Bathe, K.J., Eds., Springer-Verlag, Berlin, pp. 714-716.

Pooch, U.W. and Neider, A., 1973, "A Survey Of Indexing Techniques For Sparse Matrices," Computing Surveys, Vol. 5, No. 2, pp. 109-133.

Przemieniecki, J.S., 1968, "Theory of Matrix Structural Analysis", McGraw-Hill Book Company, New York.

Rajan, S.D. and Bhatti, M.A., 1983, "Data Management in FEM-based Optimization software," Computers and Structures, Vol. 16, No. 1-4, pp. 317-325.

RIM User's Guide, 1982, Boeing Commercial Airplane Company, P.O. Box 3707, Seattle, Washington, 98124.

Ronald, D.P., 1978, "XIO-A Fortran Direct Access Data Management System," NASA Conference Publication 2055, pp. 155-162.

Roos, D., 1966, "ICES System Design", The M.I.T. Press, Massachusetts.

Roussopoulos, N., 1979, "Tool for Designing Conceptual Schemata of Databases," Computer-Aided Design Vol. 11, No. 2, pp. 119-120.

Ryu, Y.S. and Arora, J.S., 1984, "A Study of Nonlinear Structural and Design Sensitivity Analysis Methods," Technical Report, Applied-Optimal Design Laboratory, The University of Iowa.

Schrem, E., 1978, "Functional Software Design and its Graphical Representation," Computers and Structures Vol. 8, pp. 491-502.

Shenoy, R.S. and Patnaik, L.M., 1983, "Data Definition and Manipulation Languages for a CAD Database," Computer-Aided Design, Vol. 15, No. 3, pp. 131-134.

Sobieszczanski-Sobieski, Jaroslaw, 1980, "From a Black-Box to a Programming System: Remarks on Implementation and Application of Optimization Methods," Proceedings of a NATO Advanced Study Institute Session on Structural Optimization, Sart-Tilman, Belgium.

Somekh, E. and Kirsch, U., 1979, "Interactive Optimal Design of Truss Structures," Computer-Aided Design pp. 253-258.

Southhall, J.W., 1980, "Requirements for Company-Wide Management of Engineering Information," IPAD: Integrated Programs for Aerospace-vehicle Design, NASA Conference Publication 2143, pp. 59-74.

Sreekanta Murthy, T., Reddy, C.P.D. and Arora, J.S., 1984, "Database Management Concepts in Engineering Design Optimization," Proceedings of AIAA/ASME/ASCE/AHS 25th Structures, Structural Dynamics and Material Conference.

Sreekanta Murthy, T. and Arora, J.S., 1983, "A Simple Database Management Program (DATHAN)," Technical Report, Division of Material Engg., The University of Iowa.

Sreekanta Murthy, T. and Arora, J.S., 1983, "Database Management Concepts In Design Optimization," Technical Report, Division of Material Engg., The University of Iowa.

Sreekanta Murthy, T., Reddy, C.P. and Arora, J.S., 1983, "User's Manual For Engineering Database Management System EDMS," Technical Report, Division of Material Engg., The University of Iowa.

Sreekanta Murthy, T. and Arora, J.S., 1984, "A Survey of Database Management in Engineering," Technical Report, Dept. of Civil Engineering, The University of Iowa.

Ulfsby, S., Steiner, S. and Oian, J., 1979, "TORNADO:A DBMS for CAD/CAM Systems," Computer-Aided Design, pp. 193-197.

Valle, G., 1976, "Relational Data Handling Techniques in Computer-Aided Design Process," CAD Systems, Proceedings of International Federation of Information Processing, pp. 309-326.

Vetter, M. and Maddison, R.N, 1981, "Database Design Methodology", Prentice/Hall International.

Whetstone, W. D., 1977, SPAR STructural Analysis System Reference Manual, System Level II, Vol. I, NASA CR-145098-1.

ACKNOWLEDGEMENT

This research is supported by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322.

END

FILMED

1-86

DTIC